# EditPad Pro

## Manual

Version 7.3.8 — 5 May 2015

Published by Just Great Software Co. Ltd.
Copyright © 1996–2015 Jan Goyvaerts. All rights reserved.
"EditPad" and "Just Great Software" are trademarks of Jan Goyvaerts

# Table of Contents


## EditPad Pro Manual.....1

1. File Menu.....4
2. Edit Menu.....17
3. Project Menu.....25
4. Search Menu.....38
5. Go Menu.....57
6. Block Menu.....61
7. Mark Menu.....71
8. Fold Menu.....74
9. Tools Menu.....77
10. Macros Menu.....89
11. Extra Menu.....94
12. Convert Menu.....109
13. Options Menu.....121
14. Options|Configure File Types.....133
15. Options|Preferences.....159
16. View Menu.....196
17. View|Files Panel.....206
18. View|Explorer Panel.....210
19. File Filter.....213
20. View|FTP Panel.....214
21. View|File History.....221
22. View|File Navigator.....224
23. Help Menu.....225
24. Customizing Toolbars and Menus.....237
25. Command Line Parameters.....241

## Regular Expression Tutorial.....243

1. Regular Expression Tutorial.....246
2. Literal Characters.....248
3. First Look at How a Regex Engine Works Internally.....250
4. Character Classes or Character Sets.....252
5. The Dot Matches (Almost) Any Character.....256
6. Start of String and End of String Anchors.....258
7. Word Boundaries.....262
8. Alternation with The Vertical Bar or Pipe Symbol.....265
9. Optional Items.....267
10. Repetition with Star and Plus.....268
11. Use Round Brackets for Grouping.....271
12. Named Capturing Groups.....276
13. Unicode Regular Expressions.....278
14. Regex Matching Modes.....287
15. Possessive Quantifiers.....289

16. Atomic Grouping ..... 292
17. Lookahead and Lookbehind Zero-Width Assertions ..... 294
18. Testing The Same Part of a String for More Than One Requirement ..... 298
19. Continuing at The End of The Previous Match ..... 300
20. If-Then-Else Conditionals in Regular Expressions ..... 302
21. XML Schema Character Classes ..... 305
22. POSIX Bracket Expressions ..... 307
23. Adding Comments to Regular Expressions ..... 311
24. Free-Spacing Regular Expressions ..... 312

**Regular Expression Examples ..... 313**

1. Sample Regular Expressions ..... 315
2. Matching Numeric Ranges with a Regular Expression ..... 318
3. Matching Floating Point Numbers with a Regular Expression ..... 320
4. How to Find or Validate an Email Address ..... 321
5. Matching a Valid Date ..... 324
6. Finding or Verifying Credit Card Numbers ..... 326
7. Matching Whole Lines of Text ..... 328
8. Deleting Duplicate Lines From a File ..... 330
10. Find Two Words Near Each Other ..... 331
11. Runaway Regular Expressions: Catastrophic Backtracking ..... 332
12. Repeating a Capturing Group vs. Capturing a Repeated Group ..... 338
13. Mixing Unicode and 8-bit Character Codes ..... 340

**Regular Expression Reference ..... 343**

1. Basic Syntax Reference ..... 346
2. Advanced Syntax Reference ..... 351
3. Unicode Syntax Reference ..... 355
4. Syntax Reference for Specific Regex Flavors ..... 356
5. Regular Expression Flavor Comparison ..... 358
6. Replacement Text Reference ..... 369

Part 1

# EditPad Pro Manual

# EditPad 7 Help

Welcome to the documentation for EditPad Lite 7 and EditPad Pro 7. All features that are available in EditPad Lite are also available in EditPad Pro. Some features are available in EditPad Pro but not in EditPad Lite. Those features are indicated as (available in EditPad Pro only) in this help file.

If you are upgrading from EditPad Pro 6 to EditPad Pro 7, check out what's new and learn how to migrate from EditPad Pro 6 to EditPad Pro 7.

EditPad is very configurable. Almost every aspect can be adjusted to your own tastes and habits. Many settings can be made separately for each file type. To do so, select Options|Configure File Types in the menu. To edit general preferences, use Options|Preferences.

You can access all of EditPad's functionality through the main menu. Once you get used to working with EditPad, you'll mostly rely on the fully configurable toolbar and keyboard shortcuts.

EditPad's menus are:

- File
- Edit
- Project
- Search
- Go
- Block
- Mark
- Fold
- Tools
- Macros
- Extra
- Convert
- Options
- View
- Help

Some functionality can be accessed through panels that dock to the sides of EditPad's window:

- Search and Replace
- Spell Check
- Clip Collection
- Character Map
- Byte Value Editor
- Files Panel
- Explorer Panel
- FTP Panel
- File History
- File Navigator

# 1. File Menu

## File|New

Creates a new tab with a blank, untitled file. If you click the File|New item directly, then the file type for the new file is the one for which you selected "default file type for new files" in the file type configuration. If you select a file type from the File|New submenu, then the new file uses the settings for the file type that you selected.

If you use File|Save As and give the file a name with an extension that belongs to a file type other than the one used by File|New, then the file's file type is be changed to the file type associated with that extension. The file will then also use the settings defined for that file type.

## File|Open

The File|Open command shows an open file common dialog. It allows you to open one or more files from the folder of your choice. To select more than one file, hold down the Shift or Control key on the keyboard while you click with the mouse. You can configure the initial folder of the open dialog in the Open Files Preferences.

You can tick the read-only checkbox to force EditPad to open the file in read-only mode, regardless of whether the file is writable or not. This can be useful to make sure you don't accidentally overwrite the file. It also tells EditPad not to try to open the file for writing.

If the active file is untitled and empty, it is replaced by the file you open. This ensures EditPad does not get cluttered with empty tabs.

If you open a single file that is already open, EditPad simply switches to the copy that is already open, as if you had clicked on its tab rather than attempting to open it again. If that file is open in another project, EditPad switches that project and to the file. If you open multiple files that are all open in the same project, EditPad switches to that project and to one of the files you wanted to open.

If you open multiple files and some, but not all, of them are open in projects other than the active one, EditPad moves all the files into the active project. If some of the files were open in unmanaged projects, those files are removed from those projects. If some of the files were open in managed projects, those files are closed in those projects, but remain part of the other projects.

If the active project is a managed, the File|Open command does not add the files to the project. The files show up under the project's tab as outside files. To make the files part of a managed project, either use Project|Add to Project instead of File|Open, or use Project|Add Outside Files after using File|Open.

## Recently Closed Files

The File|Open command has a submenu that lists recently closed files. Opening a file removes it from the File|Open submenu. Closing a file adds it to the top of the File|Open submenu. Selecting the file in that submenu opens it again.

Only files that were opened individually are added to the menu when they are closed. That includes files opened by double-clicking on them in Windows Explorer or by dragging and dropping them onto EditPad. Files that are opened in bulk such as by opening a project or opening a folder are not added to the File|Open menu when you close them. The Project|Open Project and Project|Open Folder commands have their own submenus for reopening projects and folders. Whether you close files one by one or all at the same time does not matter. How the file was opened determines whether it is added to the File|Open submenu.

At the bottom of the submenu, you will see the "Remove Obsolete Files" and "Remove All Files" items. The former removes all files that no longer exist from the list of recently closed files. The latter clears the list of recently closed files entirely.

Though the menu can display only 16 files, EditPad Pro actually remembers the last 100 files. To access the complete list, select the Maintain List item at the bottom. This item is only available in EditPad Pro.

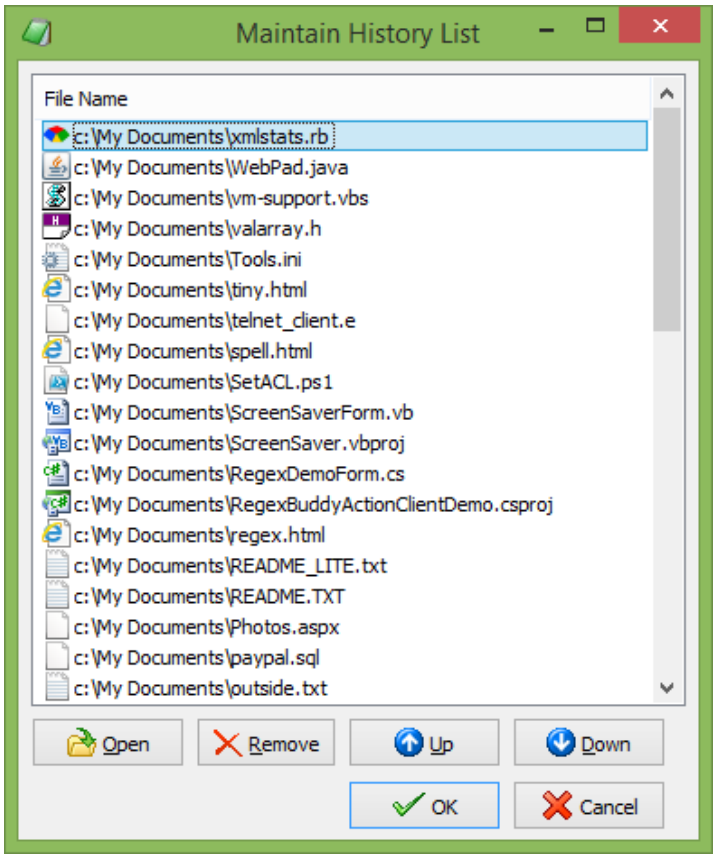
When working with projects in EditPad Pro, the Project|Add to Project has its own submenu with recently closed files. It works just like the File|Open submenu, except that it remembers the last 100 files that were closed in that project only. If you haven't worked with a project for a while, the Project|Add to Project submenu may still list closed files that have already dropped off the File|Open submenu.

## Maintain List

Though the File|Open submenu can display only 16 files, EditPad Pro actually remembers the last 100 files you closed. To access the complete list, select the Maintain List item at the bottom. This item is only available in EditPad Pro.

To open many files again at once, select all of them and click the Open button. Clicking the Open button does not close the Maintain List dialog. You can select files and click the Open button repeatedly to open multiple sets of files.

To remove files from the history, select them and click the Remove button. This will only remove files from the list. It will not delete any files. If you want to remove all files that no longer exist, you can do so by selecting "Remove Obsolete Files" in the File|Open submenu itself, rather than selecting "Maintain List" to open the above



dialog.

Using the Up and Down buttons, you can change the order of the files in the history list. Press the Control key on the keyboard while clicking the Up or Down button to move a file all the way to the top or the bottom. If you want an item to be visible directly in the File|Open submenu, move it up until it is among the first 16 files in the list.

## Read-Only Files

If a file is in use by an application, another application cannot modify it. Files stored on read-only media such as CD-ROMs cannot be modified by applications either. Files that have the read-only attribute set should not be modified by applications, but can be if you have the necessary security privileges to remove the read-only attribute.

If you use EditPad to open a file that is already in use by another application, stored on read-only media, or has the read-only attribute set, EditPad will open that file in "read only" mode. This is indicated on the status bar. Where you normally see whether the current file has been modified or not (the indicator shows "Modified" or shows nothing), you will then see "Read Only". A file that is in read only mode cannot be edited in EditPad.

You can turn off "read only" mode by clicking on the "Read Only" indicator on the status bar. If the file is in use by another application or stored on a read-only device, however, you still won't be able to save it even if you can edit it in EditPad. To save your changes, either close the other application that is using the file and then use File|Save, or save your changes to a new file with File|Save As. If the file was opened as read-only because the read-only attribute was set, EditPad will attempt to remove the read-only attribute if you try to save the file.

The File|Open screen has a "read only" checkbox at the bottom. If you tick that, EditPad opens the file as read only, even if the file is perfectly writable. In that case, clicking the "Read Only" indicator on the status bar makes the file editable, as if you had opened it without using the "read only" checkbox.

If you have a file open that is not read only and it doesn't have any unsaved changes, you can make it read-only by clicking the blank space in the status bar reserved for the "Modified" or "Read Only" indicator. This only prevents accidental changes to the file in EditPad. EditPad does not set the file's read-only attribute, nor does it prevent other applications from modifying the file.

## File|Favorites

In the Favorites submenu of the File menu, you can keep lists of files that you often work with. This way you can quickly open them to continue working on them.

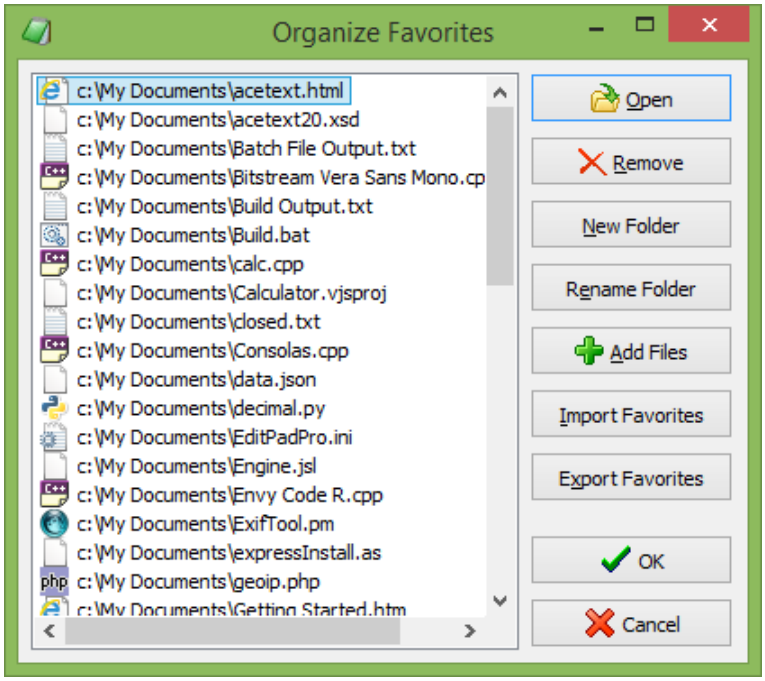
To add a file to your favorites list, first open the file in EditPad. Then select File|Favorites|Add Current File from the menu. The next time you open the File|Favorites menu, you will see the file listed there. The files are sorted alphabetically by their complete path names. If you have added folders via the Organize Favorites screen (see below), you can add a file to one of the folders by selecting the "Add Current File" item from the folder's menu.

To purge files that no longer exist from the list of favorites, select "Remove Obsolete Files" from the Favorites menu. If you have created folders, non-existent files will be purged from those folders as well. To remove only certain files, whether they still exist or not, use the Organize Favorites screen.

To open one of your favorite files, simply choose it from the Favorites menu. To open many of your favorite files, use Organize Favorites.

## Organize Favorites

Click the File|Favorites item directly or select Organize Favorites from the File|Favorites submenu to organize the list of your favorite files, or to open many of those files quickly.




Creating folders is useful if you have more than twenty favorite files. If you add more than 20 files directly to the favorites menu, or to one of the folders, then the menu listing your favorites will become very long and difficult to use. Note that creating a folder here to hold your favorites does not actually create a folder on your hard disk. It simply adds a hierarchical structure to the list of favorites stored in your EditPad Pro preferences. Use the New Folder button to create a new folder to which you can add favorites. You can change the caption of a folder by clicking on the folder and then clicking on the Rename Folder button.

When adding file to the favorites list, files are added to the main list if either nothing is selected or if a file in the main list is selected. To add files to a folder, select either the folder or a file in that folder. Click the Add Files button to find and add files.

You can move files between folders by dragging and dropping them with the mouse. EditPad Pro automatically sorts the list of favorites alphabetically. You cannot rearrange files inside a folder.

To open one or more files, select them and click on the Open button. Note that when selecting more than one file, all the files must be in the same folder. You cannot select multiple files across folders. If you want to

open files from several folders, open them folder by folder. You can click the Open button as many times as you want to open as many files as you want.

To remove one or more files from the favorites, select them and click on the Remove button. This will not delete the files, but only remove them from the list of favorites. If you want to remove all files that no longer exist, select Remove Obsolete Files, instead of Organize Favorites, from the File|Favorites menu.

## File|Templates

The Templates submenu of the File Menu works just like the Favorites submenu, with one small but important difference. When you select a file from the Favorites menu, EditPad simply opens that file. If you select a file from the Templates menu, however, EditPad will create a new, untitled file with the content of the file you selected.

If you often edit a particular file, saving the changes back into the same file, you should add it to the Favorites. If you often create new files based on the contents of a particular file, you should add that file to the Templates menu. Using templates, you won't accidentally save the changes over the original file.

The Favorites and Templates menus are completely independent. You can add the same file to both.

## File|Save

Saves the active file to disk, overwriting the original, if any, without warning.

In Save Files Preferences you can specify if and how backup copies should be created.

If the active file is untitled, File|Save invokes File|Save As.

## File|Save As

Saves the active file under a new name. If the active file has previously been saved, the original copy will remain available. If you later use File|Save after using File|Save As, it will save under the new name as well. This is indicated by the file name on the tab which changes after you use File|Save As.

When you select File|Save As, a save file common dialog box will appear. You can select the folder in which you want to save and type in a file name. You can configure the initial folder of the save dialog in the Open Files Preferences.

If you do not specify an extension for the file name, EditPad adds one depending on the selection you made in the filters (the bottommost drop-down list) in the Save As dialog box. If the active filter is "any file" (*.*), no extension is added. If any other filter is active, the default extension for that file type will be added to the file name. The default extension is the first extension listed in the Extensions setting of the file type's settings.

If you do specify an extension, EditPad will save the file with that extension. If the extension is part of another file type, all the file type dependent settings such as auto indent, number lines and word wrap will be changed to match the settings for that new file type as made in the file type configuration.

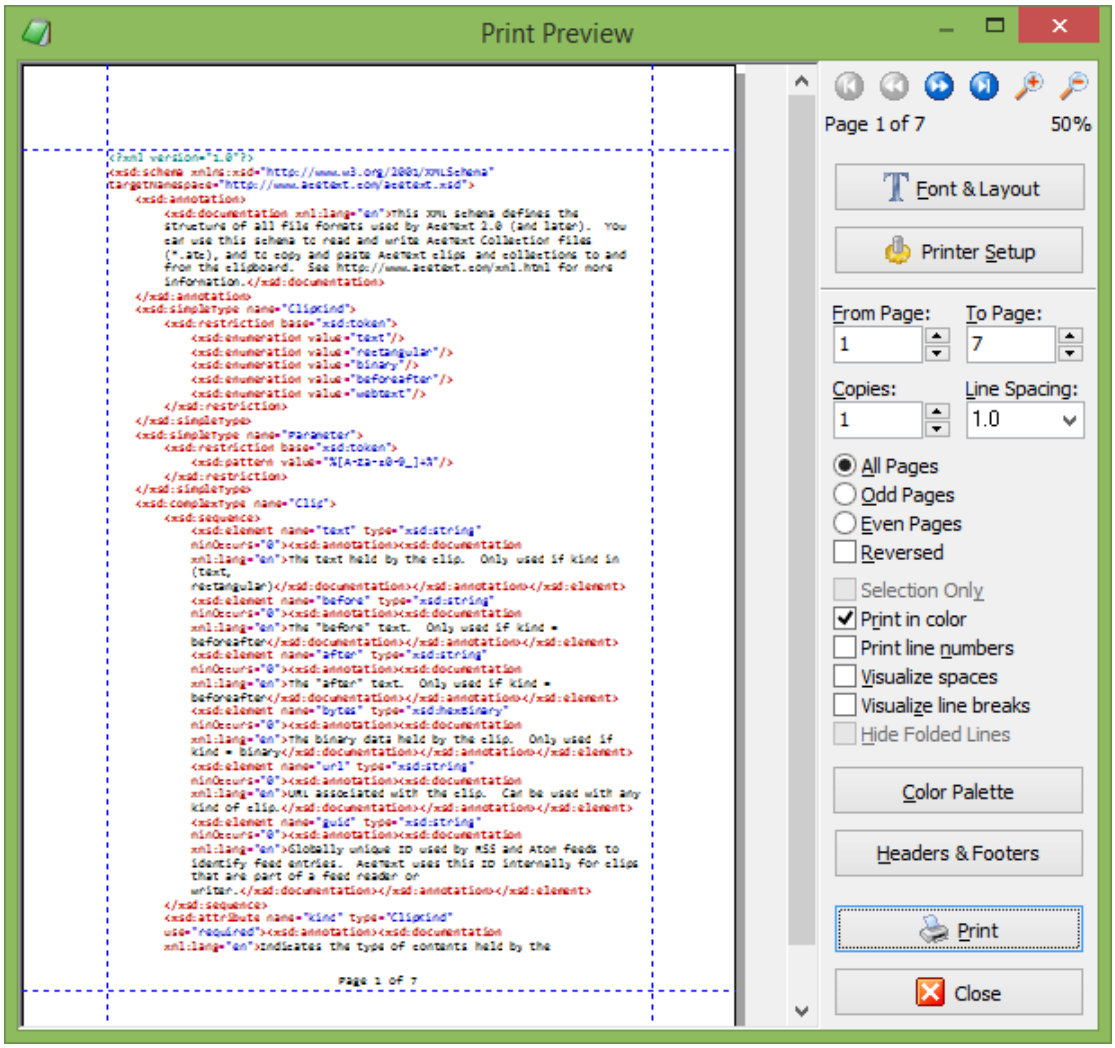
## File | Save All

Rapidly performs a File | Save for each open file, in each project.

For untitled files, File | Save As is invoked. If you click on Cancel in the save as dialog box, the save all function will not save any further files either.

## File | Print

Select File | Print in the menu to print the active file. A preview of the printout is shown first.




The margins can be changed by moving the mouse over one of the dotted blue margin lines. The mouse pointer will change to indicate that you can move the margin, and a hint box will indicate the size of the margin in centimeters and inches. Click on the line, hold the mouse button down and drag the line until it indicates the desired margin position.

With the blue arrow buttons at the top, you can browse through the pages that will be printed. You can also press the Page Up and Page Down keys on the keyboard.

The magnifying glass buttons zoom the preview in and out. You can also press the plus and minus keys on the numeric keypad on your keyboard. Zooming does not affect the printout, only the preview.

If you want to print with a different font or different text spacing, click the Font & Layout button. This button shows the same text layout configuration screen as the Options|Text Layout menu item, except that it shows the text layouts you have configured for printing rather than the ones you've configured for editing. Another difference is that if you turn on the "allow bitmapped fonts" checkbox, the font list will include printer fonts rather than screen fonts in addition to the TrueType and OpenType fonts that work everywhere. A "printer font" is a font built into your printer's hardware. If you select a printer font, set "text layout and direction" to "left to right only" for best results.

Because EditPad keeps separate text layout configurations for printing and editing, any changes you make to text layouts via the print preview only affect the printout. EditPad remembers the layout you used for printing in combination with the layout you used for editing. If you edit a file with a text layout you've called "edit monospaced" and you print it with a text layout you've called "print monospaced", then the next time you print a file you're editing with the "edit monospaced" layout, the print preview will automatically select the "print monospaced" layout.

Press the Setup button to access your printer's setup screen or to select a different printer than the default printer.

"From page" and "to page" determine the range of pages that will be printed. The value of "from page" must be smaller than or equal to that of "to page", even if you want to print in reversed order (see below). "Copies" is the number of copies that should be printed of each page.

Activating "All pages" will print all pages in the range specified by "from page" and "to page", while "odd pages" will print only the odd-numbered (1, 3, 5, ...) pages in the range, and "even pages" only the even-numbered (2, 4, 6, ...) ones. If the page range is only one page ("from page" equals "to page") and that page is odd-numbered while "even pages" is activated, or the other way around, nothing will happen when you click the Print button. You will not get an error message.

If you mark "Reversed", the pages will be printed from last to first instead of from first to last. This can be useful if you have an inkjet printer that puts pages in reversed order in the out tray.

If you opened the print preview with File|Print, you can mark the "selection only" box to print only those lines that are selected in EditPad. The "selection only" checkbox works at the level of a single line. If a line is partially selected, it will be printed. If you only want to print exactly what you selected, select Block|Print in the menu instead.

If you have a monochrome printer or you do not want to waste expensive color ink cartridges, turn off "print in color" to print black text without any background colors. If you turn on "print in colors", the printout is in full color. Be careful if you have selected the white on black color palette. You'll see white on black in the print preview, and the printout will use loads of black ink to make the whole page black.

When printing only the selection, syntax coloring may be applied differently when you use Block|Print to print the exact selection rather than using File|Print and turning on "selection only" to print the selected

lines. With Block|Print, the syntax coloring will color the block as it were the whole file, while File|Print applies syntax coloring to the whole file, even with "selection only" turned on.

Turn "print line numbers" on or off to print the file with or without line numbers, regardless of whether you've used Options|Line Numbers to edit the file with line numbers. The print preview remembers this checkbox separately for each file type.

In EditPad Pro, you can use the "visualize spaces" and "visualize line breaks" checkboxes to make spaces and line breaks visible in the printout or not, overriding the Options|Visualize Spaces and Options|Visualize Line Breaks menu items. These checkboxes are also remembered separately for each file type.

If you used the Fold|Fold command to hide certain lines, then you can turn on the "hide folded lines" option to exclude those lines from the printout. The printout will not indicate folding ranges with "plus" or "minus" buttons.

If you have "print in color" turned on, you can click the Color Palette button to select a different set of colors. You should select a palette that uses a white background. Otherwise, EditPad will use loads of ink to give the printout a colored background. If you print on colored paper, you should still select a white background in EditPad. EditPad cannot determine the color of your paper. White is assumed. The Color Palette button shows the same color selection screen as the corresponding button in the file type configuration. If you edit any palettes, those changes also affect the palettes used for editing. If you want to use different palettes for editing and for printing, create new palettes for printing instead of editing existing ones. EditPad remembers the palette used for printing in combination with the palette used for editing. E.g. if you edit a file with the "Borland classic" palette (which uses a blue background) and then select the "Embarcadero" palette for printing it without the blue background, then the print preview automatically defaults to the Embarcadero palette next time you print a file that you're editing with the Borland palette.

Click the "Print" button to start printing. Then click the "Close" button or press Escape on the keyboard to close the print preview.

## Print Headers

Click the "Headers & Footers" button in the Print Preview window to change the headers and footers to be printed, if any.

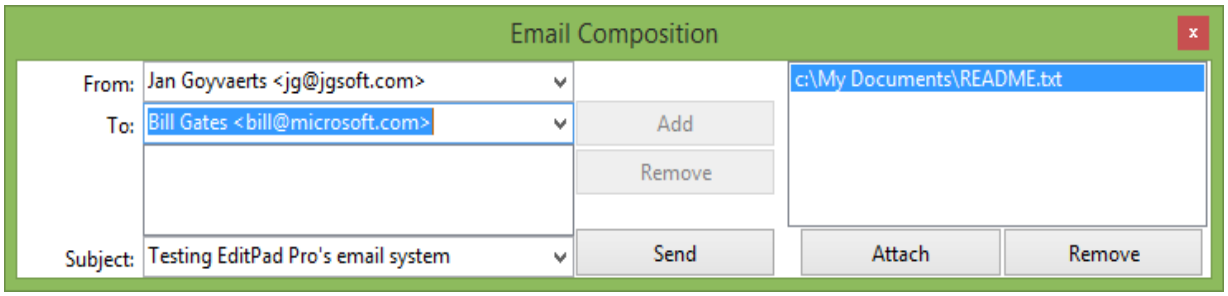
The headers and footers are printed inside the margins specified on the print preview. If no headers or footers are specified, their space will be claimed by the text body.

A few special placeholders are available in the headers and footers:

`%P` Page number of the current page
`%N` Total number of pages
`%D` Current date, printed using the short date format specified in the regional settings of the Control Panel
`%T` Current time, printed using the short time format specified in the regional settings of the Control Panel
`%FD` Date the file was last modified, again printed in short date format.
`%FT` Time of the day the file was last modified, again printed in short time format.
`%FP` Full path to the file being printed: folder + filename
`%FN` Filename (without folder name) of the file being printed

## File | Mail

Select File | Mail from the menu to send the current file to somebody via email. The email composition pane is shown.




Type in your full name followed by your own email address in the field labeled "From". If you have previously used the email function, you can quickly select your email address from the drop-down list.

Then type in the recipient's email address in the "To" field. You can either enter the recipient's full name followed by his or her email address, or you can enter only the email address.

If you want to send your message to more than one recipient, click the Add button next to the "To" field to add the first email address to the list. Then type in the second email address into the "To" field, and click the Add button again. Do so for all recipients. If you make a mistake, you can remove a recipient by selecting his or her email address from the list and clicking the Remove button.

In the "Subject" field, type in the subject of the email message.

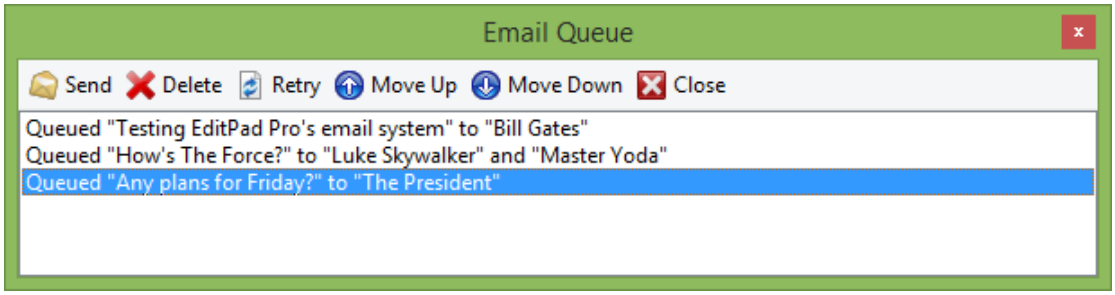
If you want to attach files to the email message, click the Attach button. Select the file you want to send. You can attach more than one file if you want to. If you change your mind about attaching a file, click on it in the list and click the Remove button below the list of attachments.

The Send or Enqueue button does not become enabled until you type in your email address, the recipient's email address, and the message's subject. When you click the Send button, the email is placed in the mail queue and sent out immediately. When you the Enqueue button, the email will be added to the mail queue, but will not be sent until you click the Send button in the mail queue. Which of the two buttons appears depends on the option to send out email immediately in the Email Preferences.

Before you can actually send mail, you need to specify the outgoing mail server that EditPad Pro should use in the Email Preferences. If you haven't done this yet, clicking the Send button brings up the Preferences screen.

## Mail Queue

The mail queue pane appears after you click the Enqueue or Send button in the mail composition pane. Each time you click the Enqueue or Send button, another email message will be added to the bottom of the mail queue. The messages will be marked as "queued".




Click the Send button in the mail queue to start sending the messages in the queue. EditPad Pro will start sending the first message in the queue marked as "queued". EditPad Pro always starts sending the first message, no matter which message you selected in the queue before clicking the Send button. When the first message has been sent, EditPad Pro continues with the next, until all queued messages have been sent. The message that is being sent is marked as "sending". After it has been sent, it will either be marked as "sent" in case of success, or "failed" in case the message could not be sent. While email is being sent, EditPad Pro tells you what is happening in the caption bar above the queue. The caption indicates "Email Queue" if you have not sent anything yet.

To remove a message from the queue, select it and click the Delete button. You can delete any message, even while EditPad Pro is sending email . Only the message that is currently being sent (if any) cannot be deleted from the queue. If you want to delete all successfully sent messages from the queue, click the Close button.

If EditPad Pro could not send a particular message, it will be marked as "failed". If you want to try the message again without making any changes to it, select it and click the Retry button. This will mark the message as "queued". You can do this whether EditPad Pro is sending messages or not. Alternatively, you can cancel sending the message by deleting it from the queue. EditPad Pro will not automatically retry failed messages. You have to mark them again as "queued".

To move a message up or down in the queue, select it and click the Move Up or Move Down button. You can move any message, regardless of its status. However, the order of the messages only matters for those marked as "queued". The first one in the list is the one that will be sent next, regardless of which message is currently being sent (if any).

Click the Close button to close the mail queue pane. This will also delete all successfully sent messages from the pane. The only way to make the mail queue appear again after closing it, is to add another message to the queue through the mail composition pane. Therefore, you cannot close the mail queue while email is being sent. If the pane takes up too much space, click on the minimize button on the caption bar below the queue. This will hide the list of messages and the buttons, but not the caption bar. This allows you to continue monitoring the progress in the caption bar, and to see the queue again by clicking the minimize button turned restore button again.

Note that the email queue sends email in the background. This means that you can continue using EditPad Pro as usual, including composing new emails, while email is being sent. This is particularly useful when sending large attachments over a slow connection. The only function you cannot use is File|Exit.

## File|Mail as Attachment

Select "Mail as Attachment" from the File menu to start with a blank email message and attach the current file to the email. The mail composition pane will appear to allow you to further prepare the email message.

## File|Reload from Disk

Select File|Reload from Disk from the menu to revert the active file to the status it had when it was last saved. This can be useful if you are viewing a file that is being written to by another program.

Note that if you switch tabs within EditPad or if you switch to another application and then back to EditPad, EditPad automatically checks whether the file on disk has been changed. If so, and you have not modified the file in EditPad, it will be automatically reloaded. If you did modify the file in EditPad, you will be asked if you want to keep the changes made in EditPad or reload from disk. In the Open Files Preferences, you can disable the automatic reload, or make EditPad always prompt before reloading.

If you click Keep Changes, EditPad will not reload the file, and will not overwrite the file on disk unless you use File|Save. If you want to keep both the changes you've made in EditPad and the modified copy of the file on disk, click the Keep Changes button, and then use File|Save As to save the changes in EditPad into a new file.

In addition, EditPad Pro gives you the option to see the difference between the file in EditPad and the file on disk. If you click the See Difference button, EditPad Pro invokes Extra|Compare with File on Disk. EditPad Pro will *not* reload the file, and will not ask you again to reload the file, until is is changed on disk again. If you want to reload the file from disk after looking at the differences, use the File|Reload from Disk menu item.

## File|Save Copy As

The Save Copy As command in the File menu works just like File|Save As. The only difference is that after File|Save Copy As, the File|Save command will continue to save the file using the original file name. The file name indicated by the tab does not change.

The Save Copy As command adds an extra "open the saved file" checkbox at the bottom of the file selection screen. If you turn on this option, EditPad opens the copy of the file that you just saved. You'll end up with two tabs, one for the original file, and one for the copy. You can edit them independently.

See File|Save As for more information on how EditPad saves files. See View|File History for a quicker way to save milestone copies of your files.

## File|Rename and Move

Use this function to move the active file into a different folder, and/or to change its file name.

This feature is similar to File|Save As, except that the original file will be deleted after the file has been saved under the new name, effectively "moving" the file.

## File|Delete

After you confirm the warning message, File|Delete closes the active file and also deletes it from disk.

If Windows keeps a Recycle Bin for the drive the active file is on, it will be moved into the Recycle Bin awaiting its final fate.

## File|Close

Closes the active file. If the file has unsaved modifications, depending on the choice you made in Open Files Preferences, EditPad will ask if you want to save, automatically save or automatically discard the changes. If you closed the last file in the project, EditPad will automatically start with a new, blank file.

You can also close the active file by clicking with the mouse wheel on the file's tab. If you have enabled the X button to be shown on tabs or the tab control in the Tabs Preferences, you can use the X button to close the file. Closing a file by closing its tab is exactly the same as using the File|Close menu item.

If the file is part of a managed project, closing open files does not remove the file from the project. Files that are part of managed projects remain part of the project as closed files. The Files Panel and Search Panel can show and search through files that are closed but are still part of managed projects. Closing outside files removes them from the project, because they were never an actual part of the project.

If you want to remove an open file from a managed project, use Project|Remove From Project to close and remove the file. If you want to remove files that you have already closed from a managed project, use Project|Remove Closed Files.

To close EditPad itself, click on the X button in the upper right of the Window, or select File|Exit in the menu.

## File|Close All

Select Close All in the File menu to close all open project and all open files. If any files have unsaved modifications, depending on the choice you made in Open Files Preferences, EditPad will ask if you want to save or not, automatically save or automatically discard the changes.

EditPad will start with a new untitled project, with one blank untitled file.

To close all files in the current project only, use Project|Close All Files, or close the project itself with Project|Close Project. To close EditPad itself, click on the X button in the upper right of the Window or select File|Exit in the menu.

## File|Close All but Current

Closes all files in the current project, except the one you're currently viewing. If any files have unsaved modifications, depending on the choice you made in Open Files Preferences, EditPad will ask if you want to save or not, automatically save or automatically discard the changes.

To close all files in all projects, except the current file, use Project|Close All but Current to close all other projects, and then File|Close All but Current to close all other files.

## File|Exit

Closes all files and terminates EditPad completely.

If you have enabled EditPad's icon next to the system clock in the System Preferences then File|Exit is not the same as clicking the X button in the upper right corner. Clicking the X button keeps EditPad in memory and keeps the icon visible, enabling EditPad to pop up instantly next time you want to edit a file. File|Exit removes the icon and shuts down EditPad completely.

If the icon next to the system clock is disabled, clicking the X button is the same as using File|Exit.

# 2. Edit Menu

## Edit|Undo

Use Edit|Undo to undo the last editing action. Repeat to undo more actions.

You can undo the undo with Edit|Redo if you do so right away. If you take any other action that is remembered by the undo function, the redo list will be cleared. Only trivial actions such as moving the cursor do not clear the redo list.

You can undo several actions in one go via the submenu of the Undo command. When you select an action, that action and all actions listed above it in the submenu will be undone at once. They will all be added to the redo list too.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box, or the replace box.

The undo feature remembers all changes you made to any file since you opened it in EditPad. Saving a file does not clear the undo list. Switching between files switches between the undo lists of those files. EditPad remembers the changes you made to each file even while you work with other files, until you close each file.

Undoing and redoing actions updates the file's "modified" status as indicated by the status bar and tab color. When you undo all changes you made since last saving the file, or redo all changes you undid since last saving the file, the file will be indicated as being unmodified.

EditPad may use quite a lot of memory to remember all the changes you made to all the files you have open. Some commands, such as a search-and-replace across all files, may result in very large numbers of changes. EditPad's undo history includes a safeguard to make sure EditPad does not run out of memory. If the undo history grows too large, EditPad automatically discards the oldest changes. In extreme cases, like a search-and-replace that makes millions of replacements, the undo history may be cleared entirely.

In EditPad Lite, the memory limits are set automatically to make sure EditPad Lite doesn't use up all of your computer's memory, while still being able to keep a complete undo history. Unless you're making millions of changes in a search-and-replace, EditPad Lite's undo history can easily keep track of a full day's worth of text editing.

In EditPad Pro, you can configure the limits in the System Preferences. The default limits use the same balance as EditPad Lite. The actual numbers depend on the amount of RAM your PC has.

In practical terms, you needn't worry about EditPad's undo history. It'll remember virtually everything, without causing your computer to run out of memory and crash. If you do need to undo very long editing sessions, EditPad's backup options and File History may be more practical anyway.

## Edit|Redo

If you change your mind after undoing an action with Edit|Undo, use Edit|Redo to redo it.

The redo function works exactly the same like the undo function, except that making any change to a file in any way except through Edit|Undo or Edit|Redo, will clear the redo list.

In contrast with other text editors, EditPad Lite and Pro do not clear the redo list when you take a trivial action such as moving the text cursor or making a selection. This gives you more opportunity to change your mind and redo undone actions.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Edit|Cut

Deletes the current selection and places it on the Windows clipboard. Use Edit|Copy to put the selection on the clipboard without deleting it. If there is no current selection, and the option to copy the active line is turned on in the Editor Preferences, then the entire current paragraph is deleted and placed onto the clipboard.

Any data previously held by the clipboard is removed from it. Use Edit|Cut Append instead if you do not want this.

EditPad always copies text to the clipboard using Windows line breaks. This ensures maximum compatibility with other Windows applications. If the file uses syntax coloring, then EditPad places both plain text and rich text on the clipboard. When you paste the rich text version into a word processor, it will be pasted with the same font and colors you were using in EditPad. The rich text version is only generated when another application requests it. When copying and pasting within EditPad or between EditPad and another plain text editor, no time or memory is wasted to generate the RTF. If you don't want word processors to paste rich text, either use the "paste as plain text" or "paste unformatted text" command in your word processor, or tell EditPad Pro not to copy rich text in the Editor Preferences.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

If you're editing a file in hexadecimal mode, and you cut something from that file, and you paste that into another application or into a file you're editing in text mode in EditPad, the result will differ depending on whether the cursor was in the hexadecimal section or the ASCII section when you used Edit|Cut. If you cut from the hexadecimal section, and paste into another application, the other application will see the hexadecimal representation on the clipboard, e.g.: "74657374". If you cut from the ASCII section, the other application will see the actual bytes, e.g.: "test".

If you cut and paste between EditPad and another hex editing application, you may need to switch EditPad to hexadecimal mode and place the cursor in the hexadecimal or ASCII section of the editor to get the results you expect.

## Edit|Cut Append

Deletes the current selection and places it on the Windows clipboard. Use Edit|Copy Append to do this without deleting the selection from EditPad. If there is no current selection, and the option to copy the active line is turned on in the Editor Preferences, then the entire current paragraph is deleted and placed onto the clipboard.

If the clipboard already holds plain text data, the selection is appended to the text already on the clipboard, even if the text was placed on the clipboard by another application. Edit|Paste will then paste both the original text plus the selection appended to it.

If the file uses syntax coloring, and the clipboard already holds a rich text version of other text copied by EditPad, then the rich text version is appended to as well. If the clipboard holds rich text copied by another application, Edit|Cut Append removes it from the clipboard.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

## Edit|Copy

Places the current selection on the Windows clipboard. If there is no current selection, and the option to copy the active line is turned on in the Editor Preferences, then the entire current paragraph is placed onto the clipboard.

Any data previously held by the clipboard is removed from it. Use Edit|Copy Append instead if you do not want this.

If you select a block that includes one or more folded sections and copy the block to the clipboard, all selected lines, including lines hidden by folding, will be copied to the clipboard. Use Fold|Copy Visible Lines if you don't want to copy the hidden lines. When pasting back into EditPad Pro, the folded sections remain folded.

EditPad always copies text to the clipboard using Windows line breaks. This ensures maximum compatibility with other Windows applications. If the file uses syntax coloring, then EditPad places both plain text and rich text on the clipboard. When you paste the rich text version into a word processor, it will be pasted with the same font and colors you were using in EditPad. The rich text version is only generated when another application requests it. When copying and pasting within EditPad or between EditPad and another plain text editor, no time or memory is wasted to generate the RTF. If you don't want word processors to paste rich text, either use the "paste as plain text" or "paste unformatted text" command in your word processor, or tell EditPad Pro not to copy rich text in the Editor Preferences.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

If you're editing a file in hexadecimal mode, and you copy something from that file, and you paste that into another application or into a file you're editing in text mode in EditPad, the result will differ depending on whether the cursor was in the hexadecimal section or the ASCII section when you used Edit|Copy. If you copy from the hexadecimal section, and paste into another application, the other application will see the

hexadecimal representation on the clipboard, e.g.: "74657374". If you copy from the ASCII section, the other application will see the actual bytes, e.g.: "test".

If you copy and paste between EditPad and another hex editing application, you may need to switch EditPad to hexadecimal mode and place the cursor in the hexadecimal or ASCII section of the editor to get the results you expect.

## Edit|Copy Append

Places the current selection on the Windows clipboard. If there is no current selection, and the option to copy the active line is turned on in the Editor Preferences, then the entire current paragraph is placed onto the clipboard.

If the clipboard holds text data, the selection is appended to the text already on the clipboard. Edit|Paste will then paste both the original text plus the selection appended to it.

If the file uses syntax coloring, and the clipboard already holds a rich text version of other text copied by EditPad, then the rich text version is appended to as well. If the clipboard holds rich text copied by another application, Edit|Copy Append removes it from the clipboard.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

## Edit|Paste

If you select Edit|Paste when the Window clipboard holds textual data, it will be inserted into the active file at the current position of the text cursor. If EditPad is in overwrite mode, and you're not pasting whole lines (see below), the pasted text overwrites the text after the cursor, as if you had typed in the text. Pressing the Insert key on the keyboard or clicking the Insert/Overwrite indicator on the status bar toggles between insert and overwrite modes.

If the active file does not use Windows line breaks, the text on the clipboard is automatically converted to the correct line break style before it is inserted. The text on the clipboard remains untouched, in case you want to paste it into other applications too.

If you want to paste a block of text from another application as a rectangular selection, first make a rectangular selection in EditPad, and then paste. EditPad will then interpret the text on the clipboard as a rectangular block and replace the selection with it. Text copied from EditPad Pro is always pasted in the way (linear or rectangular) you had it selected when you cut or copied it.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

In EditPad Pro, the option "paste whole lines when lines are copied as a whole" affects how text is pasted if you copied complete lines to the clipboard. Copying a complete line means to copy everything from the start of the line to the end of the line, including the line break at the end of the line. Copying multiple lines completely means copying everything from the start of the first line in the block until the end of the last line

in the block, including the line break at the end of the last line. When the option "paste whole lines when lines are copied as a whole" is on, lines that were copied as a whole are always pasted as if the cursor were at the start of the line when you're pasting. Thus, lines copied as a whole are always pasted as a whole before the line that the cursor is on when pasting. This makes it easy to move blocks of lines around without worrying about the horizontal position of the text cursor. If this option is off, text is always pasted at the exact spot the text cursor is at, even when whole lines were copied. Only whole lines copied in EditPad can be pasted as whole lines. EditPad cannot determine whether text copied from other applications is a whole line or not. EditPad Lite does not have this option and always pastes at the exact spot the text cursor is at.

In hexadecimal mode, if the clipboard contains a hexadecimal representation of characters, the effect of the Paste command depends on whether the text cursor is in the hexadecimal section at the left, or the ASCII section at the right. If you paste into the hexadecimal section, EditPad will paste the bytes represented by the text on the clipboard. If you paste into the ASCII section, EditPad will paste the text on the clipboard "as is". E.g. if the clipboard holds "74657374", pasting into the hex section inserts "test", while pasting into the ASCII section inserts "74657374" into the file.

If the text on the clipboard is not text with hexadecimal values, EditPad pastes the text regardless of whether the cursor is in the text or hexadecimal section. If the clipboard holds something you copied in hexadecimal mode in EditPad, EditPad will always paste the bytes that you copied, even if you moved the cursor from the ASCII to hex section or vice versa. If you copied text from a file in EditPad that you're editing in text mode, then the hex interpretation does take place.

If you copy and paste between EditPad and another hex editing application, you may need to switch EditPad to hexadecimal mode and place the cursor in the hexadecimal or ASCII section of the editor to get the results you expect.

## Edit|Swap with Clipboard

If you select Edit|Swap with Clipboard when the Window clipboard holds textual data, it will be swapped with the current selection in the active file. That is, the selection is put on the clipboard as happens when you use Edit|Cut and then the text previously held by the clipboard is pasted into the text like Edit|Paste does.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

## Edit|Select All

Selects all the text in the active editor. This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

## Edit|Insert Date & Time

Inserts the current date and time at the position of the text cursor. If you click this item directly or use its keyboard shortcut, the date and/or time are inserted using the format you last used. If you've never used this command before, it defaults to the short date and time formats set in the Windows Control panel.

To use a different date and time format, or to insert the date only or the time only, select Other Date & Time format in the submenu of the Insert Date & Time menu item. A screen will pop up with a list of date and time specifiers. You can type in the date and/or time format that you want in the edit box at the top. You can double-click a specifier in the list to add it to the format. The "actual date and time" box shows a preview of the text that will be inserted when you click OK.

The Insert Date & Time submenu keeps a history of the last 16 formats that you used. Click on one to insert the current date and/or time using that format.

## Date and Time Format Specifiers

These are the date and time placeholders that you can use. Format specifiers may be written in upper case as well as in lower case letters. Both produce the same result.

`c` Displays the date in short format followed by the time in long format as specified in the regional settings in the Windows Control Panel.
`d` Displays the day as a number without a leading zero (1-31).
`dd` Displays the day as a number with a leading zero (01-31).
`ddd` Displays the day as an abbreviation (Sun-Sat) in the language of the current Windows locale.
`dddd` Displays the day as a full name (Sunday-Saturday) in the language of the current Windows locale.
`ddddd` Displays the date using the short date format specified in the regional settings in the Windows Control Panel.
`dddddd` Displays the date using the long date format specified in the regional settings in the Windows Control Panel.
`m` Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
`mm` Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
`mmm` Displays the month as an abbreviation (Jan-Dec) in the language of the current Windows locale.
`mmmm` Displays the month as a full name (January-December) in the language of the current Windows locale.
`yy` Displays the year as a two-digit number (00-99).
`yyyy` Displays the year as a four-digit number (0000-9999).
`h` Displays the hour without a leading zero (0-23).
`hh` Displays the hour with a leading zero (00-23).
`n` Displays the minute without a leading zero (0-59).
`nn` Displays the minute with a leading zero (00-59).
`s` Displays the second without a leading zero (0-59).
`ss` Displays the second with a leading zero (00-59).
`z` Displays the millisecond without a leading zero (0-999).
`zzz` Displays the millisecond with a leading zero (000-999).
`t` Displays the time using the short time format specified in the regional settings in the Windows Control Panel.
`tt` Displays the time using the long time format specified in the regional settings in the Windows Control Panel.
`am/pm` Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.

`a/p` Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
`/` Displays the date separator character specified in the regional settings in the Windows Control Panel.
`:` Displays the time separator character specified in the regional settings in the Windows Control Panel.
`'xx'/"xx"` Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

## Edit|Insert Matching Bracket

The Insert Matching Bracket command is only available when bracket matching is enabled in the syntax coloring section of the file type configuration for the active file's file type. The syntax coloring scheme determines exactly which brackets are matched. See that section in this help file to learn how bracket matching works in EditPad Pro.

Just like the syntax coloring scheme determines which brackets are matched, it also determines what is inserted by the Insert Matching Bracket command. When an unclosed opening bracket is highlighted, the corresponding closing bracket is inserted. When an unmatched closing bracket is highlighted, the corresponding opening bracket is inserted. The HTML and XML schemes insert matching opening and closing tags.

The Insert Matching Bracket command is intended to be used while typing with minimal cursor movement. If you wanted to type "this bold word" into an HTML file, you would first type "this bold", then use Edit|Insert Matching Bracket, and then type " word". When you use Edit|Insert Matching Bracket, the cursor is automatically placed after the inserted tag.

Do not use Insert Matching Bracket immediately after typing "this ". Though it may seem neat to close the bold tag immediately, before typing its contents, this requires unnecessary cursor movement. You'd end up with the cursor after "this " requiring you to move the cursor back over the "" tag, then type "bold", then move the cursor forward over "" and then type " word". Since EditPad Pro automatically highlights the "" tag as long as it is unclosed, there is no need to close it immediately. Simply leave it unclosed until you have typed in everything you want between the tag, and then use Edit|Insert Matching Bracket.

If you use Insert Matching Bracket when an unmatched closing bracket is highlighted, it inserts the matching opening bracket. The cursor is then placed to the left of the bracket that was inserted. This way you can easily use Edit|Insert Matching Bracket repeatedly to insert multiple opening brackets. E.g. if you type `"unclosed)}]"` into a C++ file, then put the cursor before "unclosed", and then use Edit|Insert Matching Bracket three times, you'll get `"[{(unclosed)}]"`.

The Insert Matching Bracket command can match up an unmatched bracket even when matching brackets are highlighted. In the C statement `if (a == b) && (c == d))` the last `)` is unmatched. If you put the cursor before the first `(`, EditPad Pro will highlight it as being matched with the `)` after `b`. This clearly shows that the `if` statement isn't properly enclosed in parentheses. If you use Edit|Insert Matching Bracket with the cursor at this position, EditPad Pro looks for the nearest unmatched bracket. In this case that is the final `)` in the statement. EditPad Pro inserts `(`, correctly completing `if ((a == b) && (c == d))`.

## Edit|Duplicate Line

Duplicates the line the text cursor is on.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Edit|Insert Page Break

Select Insert Page Break from the Edit menu or press Ctrl+Enter to insert a page break into the current file at the current position of the text cursor.

A page break will show up as a horizontal line while you edit the file in EditPad Pro. When you print the file, a new page will be started at the page break. The first paragraph below the horizontal line indicating the page break, will be the first paragraph on the new page.

## Edit|Delete

Deletes the selected part of the current file. If nothing is selected, or when selections are persistent, the character to the right of the cursor is deleted.

This command is always invoked on whichever editor is showing the text cursor (vertical blinking bar), regardless of whether you use the main menu, a toolbar button, or a keyboard shortcut.

## Edit|Delete Line

Selects the line the text cursor is currently on, and deletes it. If the cursor is on the first line of a folded block of lines, indicated by a square with a + in it in the left margin, then all the lines folded underneath the current line are also deleted.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

# 3. Project Menu

## About EditPad Pro Projects

An EditPad Pro Project is a series of files that you can open all at once through Project|Open Project. If you regularly work with a set of related files, it is convenient to create a project file for them. You could have a project for your web site, for the source code of an application you are working on, the chapters of your next book, etc.

When you start EditPad Pro, it opens with an untitled project containing one untitled blank file. To create a project, simply open all the files that should be in the project. If you already have some files open that you don't want to be in the new project, select Project|New Project in the menu before opening the files. After opening the files, save the project with Project|Save Project As. From that point on, EditPad Pro automatically maintains the project. Whenever you open new files into the project, or close files, the project's file list is updated.

If you prefer to actively manage the list of files in the project, turn on the Managed Project option in the Project menu. Then you can use the commands below that item to add files to the project or remove them. You can still open and close files as usual using the File menu, but opened files won't be added to the project, and closed files won't be removed.

Project files also keep track of the last editing position, bookmarks and editing options for each file in the project. When you reopen a project the next time you start EditPad Pro, it will be in exactly the same state as it was in when you last closed the project. EditPad Pro will not bother you with prompts to save the project.

You can have as many projects open at the same time as you want. Creating a new project or opening an existing one does not close the files or projects that you already have open. By default, once you have two or more projects open, a second row of tabs will appear above the row of tabs for files. The new row of tabs allows you to switch between open projects. The tabs for files will list only those files that are part of the active project. Tabs can be configured in the Tabs Preferences.

## Project|New Project

Select New Project in the Project menu to start with an untitled project containing a blank, untitled file.

An untitled project is simply a holding space for files. When working with lots of files at the same time, it's often handier to create a bunch of untitled, temporary projects to hold different sets of files. Instead of having one long row of tabs for all the files, you'll have one row of tabs for the projects, and one row of tabs showing only the files in the active project. It's often faster to switch between files by clicking on one project tab and on one file tab, than to scroll through a long list of file tabs.

If you regularly work with the same set of files, save the project with Project|Save Project As. From that point on, EditPad Pro automatically maintains the project. Whenever you open new files into the project, or close files, the project's file list is updated.

If you prefer to actively manage the list of files in the project, turn on the Managed Project option in the Project menu. Then you can use the commands below that item to add files to the project or remove them.

You can still open and close files as usual using the File menu, but opened files won't be added to the project, and closed files won't be removed.

## Project|Open Project

Opens an EditPad Pro Project saved with Project|Save Project. Click the Project|Open Project item directly to show a file selection window from which to select the `.epp` file. Or use the submenu of the Project|Open Project item to reopen a recently closed project. Opening a project removes it from the Project|Open submenu. Closing a project adds it to the top of the Project|Open submenu. You can manage the Project|Open Project submenu just like the File|Open submenu.

You can have as many projects open at the same time as you want. Opening a project does not close any projects that are already open. If you opened individual files before opening the project, those files are in an untitled project that will remain open. If the active project is untitled and holds only a blank untitled file, then the blank project is closed when you open a project.

Each file can be open only once in a single EditPad instance. If the project you're opening holds a file that is already open in another project that is open in EditPad, the file is closed in the other project. If that project is unmanaged, that causes the file to be removed from the other project. If the other project is managed, the file is still closed in that project, but it remains part of it. The file's options, cursor position, and any unsaved changes are all preserved. The file is closed in the other project and moved "as is" to the newly opened project.

## Project|Favorite Projects

The Project|Favorite Projects menu works just like the File|Favorites menu, except that it lists EditPad Pro project files rather than text files. You have to save a project before you can add it to the favorites.

## Project|Save Project As

Select Save Project As from the Project menu to save the current project under a different name. From that point on, EditPad Pro will automatically maintain the project, updating the file you saved it into. Whenever you open new files into the project, or close files, the project is updated. If the project was previously saved under another name, the old project file will not be updated unless you reopen the old project.

You only need to use Save Project As when you want to save an untitled project, or you want to create a new copy of a previously saved project. There is no need to repeatedly save projects under the same name to prevent losing changes like you would do with files. EditPad Pro always saves project files automatically when the state of the project changes as soon as you've saved the project once to give it a file name.

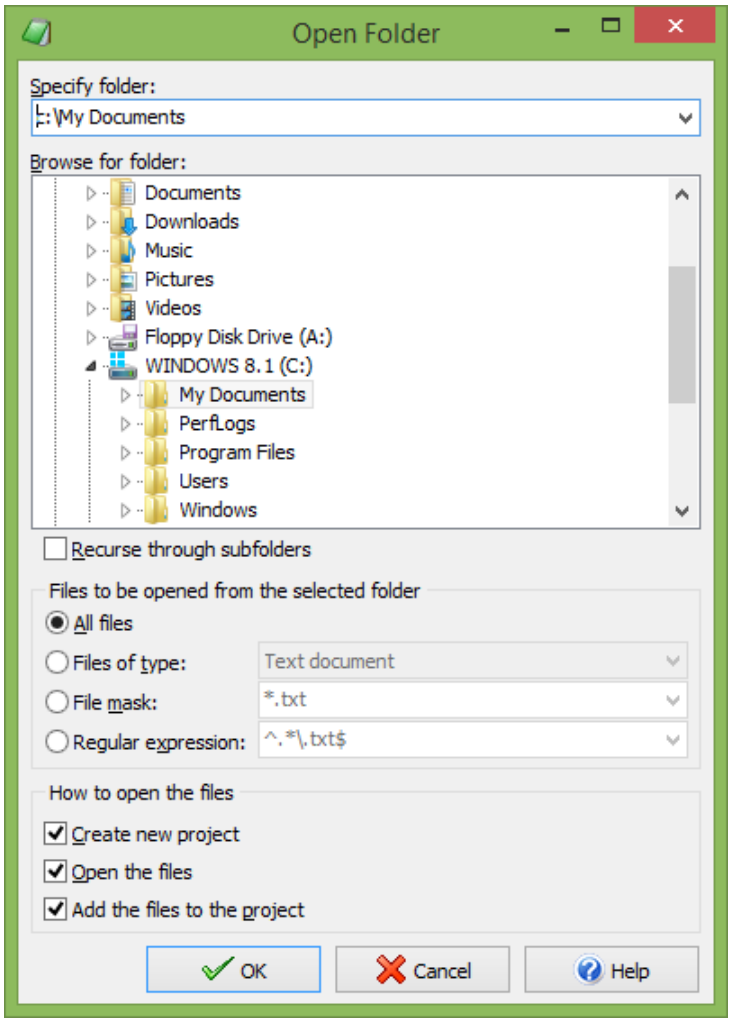
## Project|Save All Files in Project

Rapidly performs a File|Save for each open file in the current project.

For untitled files, File|Save As is invoked. If you click on Cancel in the save as dialog box, the save all function will not save any further files either.

## Project|Open Folder

Select Open Folder from the Project menu to open all files or files of a certain type from a particular folder.




You can type in the path to the folder containing the files you want to open into the drop-down list at the top. If you have recently opened files from this folder, you can select it from the drop-down list. EditPad Pro remembers the last 16 folders. Another way to select the folder is to browse for it using the folder tree. The drop-down list and folder tree are automatically kept in sync.

Mark the option "recurse through subfolders" if you also want to open files contained in subfolders of the folder you selected.

If you want to open all files in the folder (and optionally its subfolders), select the "all files" option. If you want to open only files from a certain type, you can select "files of type" and pick one of the file types from

the drop-down list. The list will contain all the file types you defined in Configure File Types. This will open all files that match the file type's file mask.

To open files matching another list of extensions, select "file mask" and type in the list of extensions or file masks, separated by semicolons, into the "file mask" drop-down list. You can also select a recently used file mask from the drop-down list.

If you choose the "regular expression" option, then the regular expression must find a (partial) match in the name of a file for it to be included in the search. This regex is used to filter files by their names only. If you want to search through the contents of the files, use the Find on Disk command in the Search menu instead of the Open Folder command.

Tick "create new project" to open the files into a new, untitled project. This is the same as using Project|New Project before using Project|Open Folder.

Tick "open the files" to make EditPad Pro actually open all of the files. If you turn this off, then "add files to the project" is automatically turned on, and EditPad Pro adds the files to the project without opening them. The files will show up in the Files Panel as closed files. Since only managed projects can contain closed files, turning off "open the files" automatically changes the current or the new project that the files are added into a managed project.

Tick "add the files to the project" to add the files to the project. If you turn this off, then "open the files" is automatically turned on, and the files will be opened as outside files, without adding them to the project. Since only managed projects can have files open that aren't part of the project, turning off "add the files to the project" automatically changes the current or the new project that the files are opened into a managed project.

## Recently Opened Folders

The Open Folder item has a submenu that lists recently opened folders. Each time you use Project|Open Folder, the opened folder is added to top of this submenu. Though the menu shows only the folder's path, all the settings from the Open Folder window are remembered by the menu. Select Maintain List in the submenu to see the settings used for each folder. When you select one of the remembered folders, EditPad Pro opens the files contained by that folder, using the same file mask and other options as you last used for that folder. If you have opened files from this folder more than once, the folder will appear only once in the Reopen Folder menu, with the most recently used settings.

"Remove Obsolete Folders" removes all folders that no longer exist from the Open Folders submenu. EditPad Pro does not check whether the folders still contain any files matching the file mask. EditPad Pro only checks whether the folders themselves still exist. "Remove All Folders" clears the Open Folders submenu.

## Project|Managed Project

New projects created with Project|New Project are always unmanaged projects. The blank project that EditPad Pro starts up with is also unmanaged. If you used EditPad Pro 6 in the past, projects in EditPad Pro 6 were always unmanaged.

In EditPad Pro 7, an **unmanaged project** means that you won't be explicitly managing which files are part of the project and which aren't. The project simply consists of the files that you have open in the project. Opening a file automatically adds it to the project, and closing a file automatically removes it from the project.

Unmanaged projects are a great way to keep groups of files that you are working with, without requiring any extra effort from you. Simply save the project once to give it a file name. From then on, EditPad Pro automatically updates the project. The project will always open the set of files you had open in it last time.

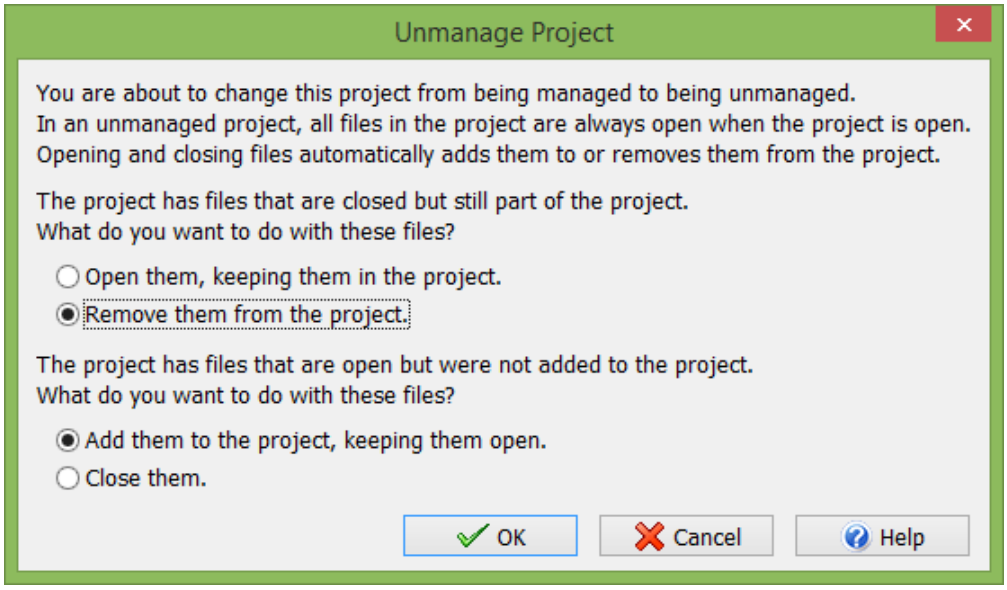
A **managed project** means that EditPad Pro expects you to manage which files are part of the project, and which aren't. A managed project can contain three kinds of files: **open files** are part of the project and are in EditPad; **closed files** are part of the project, but are not open in EditPad; **outside files** are not part of the project, but are open under the project's tab in EditPad. If you close a managed project and then open it again, the "open files" are opened again, the "closed files" are not opened but remain part of the project, and the outside files are gone (they may still exist as separate files, but the project won't remember them). Open files and outside files have their own tabs under the project's tab. Closed files do not have tabs until you open them again.

In the Files Panel, all files in unmanaged projects, and open files in managed projects are listed using a plain font. Outside files in managed projects are listed in italics. Closed files in managed projects are listed with a dimmed font. Closed files may not be listed at all if you turned off the option to show closed files in the Files Panel. Double-clicking a closed file opens it.

Managed projects take a bit more effort. It's worth it for projects that consists of large sets of files because a managed project doesn't force you to open all the files that it contains. You can quickly access closed files from the Files Panel and you can even search through closed files without opening them (as long as the project is open). Managed projects are also great for preventing accidental changes. Files are opened as outside files, and closing files doesn't remove them. The list of files in the project remains the same, unless you explicitly add files to the project or remove files from the project.

To turn an unmanaged project into a managed one, select the Managed Project item in the Project menu or click the corresponding toolbar button. The icon on the project's tab changes from the red unmanaged project icon to the blue managed project icon. All files that were open in the unmanaged project become part of the managed project as open files.

You can turn a managed project back into an unmanaged one by selecting the Managed Project item in the Project menu again. If the managed project consisted of open files only, all that happens is that the icon on the project's tab changes from blue to red to indicate the project is unmanaged. If the managed project contained closed files and/or outside files, EditPad Pro will ask what you want to do with those files.


Unmanage Project
You are about to change this project from being managed to being unmanaged.
In an unmanaged project, all files in the project are always open when the project is open.
Opening and closing files automatically adds them to or removes them from the project.
The project has files that are closed but still part of the project.
What do you want to do with these files?
Open them, keeping them in the project.
Remove them from the project.
The project has files that are open but were not added to the project.
What do you want to do with these files?
Add them to the project, keeping them open.
Close them.
OK
Cancel
Help

Unmanaged projects cannot contain closed files. If you try to turn a managed project that contains closed files into an unmanaged project, you need to choose what to do with the closed files. If you select to open them, the closed files will be opened and remain part of the unmanaged project as open files. If you select to remove them, the closed files are removed from the project. They will remain as individual files on your hard disk.

Unmanaged projects cannot have outside files. If you try to turn a managed project that has outside files into an unmanaged project, you need to choose what to do with the outside files. If you select to keep them open, the outside files remain open and will be added to the unmanaged project as open files. If you select to close them, the outside files are closed and removed from the project. They will remain as individual files on your hard disk.

The project menu offers several commands that you can use to handle open files, closed files, and outside files in managed projects:

Add to Project: Select files from disk to be opened and added to the project as open files.
Add to Project Unopened: Select files from disk to be added to the project as closed files.
Add Active File: If the active file is an outside file, add it to the project as an open file.
Add Outside Files: If the project has one or more outside files, add all of them to the project as open files.
Open Closed Files: If the project contains one or more closed files, open them all.
Remove From Project: Close the active file and remove it from the project.
Remove Closed Files: If the project contains one or more closed files, remove them all from the project.
Close Outside Files: If the project has one or more outside files, close them all and remove them from the project.
Close All Files: Close all files in the project. If the project has outside files, those are removed from the project. Open files remain part of the project as closed files.

## Project|Add to Project

The Add to Project command in the Project menu works like the File|Open command with two small but important differences.

If the active project is a managed project, files opened with Project|Add to Project are opened and added to the project as open files. Files opened with File|Open are opened as outside files that are not part of the project. If you close the managed project and then reopen it, files opened with Project|Add to Project are opened again along with the project, while files opened with File|Open are not remembered by the project.

If you have already opened files with File|Open and they now appear as outside files in your managed project, you can use Project|Add Outside Files to make them part of the project.

If the active project is an unmanaged project, then there is no difference between opening files using File|Open or Project|Add to Project. Both commands add the files and open files to unmanaged projects.

The other difference is the submenu with recently closed files. This applies to both managed and unmanaged projects. The File|Open submenu lists files that were opened and then closed in any project, including untitled projects that were never saved. This list is saved by EditPad along with its general settings and history information. The Project|Add to Project submenu lists files that were opened and then closed in the current project only. This list is saved in the `.epp` project file.

If you haven't worked with a particular project for a while, any files that you had open in that project and then closed will have long dropped off the 100 files that the File|Open submenu remembers. But when you open that project again, the Project|Add to Project submenu will still remember the last 100 files you closed while working with that project.

The File|Open and Project|Add to Project submenus remember all files that were closed even if you didn't use that menu item to open them. For example, if you double-click on a file in Windows Explorer and then close the file, it will be listed at the top in both File|Open and Project|Add to Project. If you then switch to another project, the file will still be at the top in File|Open but won't be in the Project|Add to Project submenu for the project you switched to. Files that are opened in bulk such as by using Project|Open Folder are not remembered by either menu.

## Project|Add to Project Unopened

The Add to Project Unopened command in the Project menu shows a file selection window just like File|Open and Project|Add to Project. Unlike these two commands, Add to Project Unopened does not open any files. The selected files are added to the project as closed files.

Only managed projects can contain closed files. If the active project is unmanaged, using the Add to Project Unopened command automatically turns the project into a managed one, as if you had selected Project|Managed Project in the menu.

## Project|Add Active File

The Add Active File command in the Project menu is only available if the active project is managed and the active file is an outside file. An "outside file" is a file that is open under the project's tab, but is not actually part of the project. When a project is closed and reopened, outside files are not remembered. Use the Add Active File command to make the file a part of the project, so it will be remembered by the project.

If the active project has many outside files, you can use Project|Add Outside Files instead if you want to make them all part of the project.

## Project|Add Outside Files

The Add Outside Files command in the Project menu is only available if the active project is managed and it has one or more outside files. An "outside file" is a file that is open under the project's tab, but is not actually part of the project. When a project is closed and reopened, outside files are not remembered. Use the Add Outside Files command to make all such files part of the project, so they will be remembered by the project.

If you want to add some of the outside files but not all of them, use the Project|Add Active File to add them one by one.

## Project|Open Closed Files

The Open Closed Files command in the Project menu is only available if the active project is managed and it has one or more closed files. A "closed file" in a managed project is a file that was closed but is still part of the project. The Open Closed Files command opens all such files in the active project.

If you want to open some, but not all, closed files in a managed project, use the Files Panel. If you want to search through all files in a managed project, there is no need to open closed files. Simply turn on the Closed Files search option to search through all files that are part of the project, whether they are open or closed.

The Open Closed Files command is not available for unmanaged projects, because closing a file in an unmanaged project automatically removes it from the project. For managed projects, the Open Closed Files command does not reopen files that were removed from managed projects with Project|Remove From Project and does not reopen outside files that were not part of the project when they were closed. Such files may still be listed in the Project|Add to Project submenu. You can use that submenu to reopen them.

## Project|Remove from Project

The Remove from Project command in the Project menu closes the active file and removes it from the project. For unmanaged projects, there is no difference between Project|Remove from Project and File|Close. For managed projects, files closed with Project|Remove from Project are completely removed from the project, while files closed with File|Close remain part of the project if they were added to the project.

If you want to remove a file that is already closed from a managed project, use the Files Panel. If you want to remove all closed files from a managed project, use Project|Remove Closed Files. If you want to remove all outside files from a project, use Project|Close Outside Files.

Files that are removed from a project are added to the File|Open and Project|Add to Project submenus that list recently closed files, unless the files were opened in bulk withProject|Open Folder or a similar command.

## Project|Remove Closed Files

The Open Closed Files command in the Project menu is only available if the active project is managed and it has one or more closed files. A "closed file" in a managed project is a file that was closed but is still part of the project. The Remove Closed Files command removes all such files from the the active project.

If you want to remove only some closed files from the project, use the Files Panel.

Files that are removed from a project are added to the File|Open and Project|Add to Project submenus that list recently closed files, unless the files were opened in bulk withProject|Open Folder or a similar command.

## Project|Close Outside Files

The Close Outside Files command in the Project menu is only available if the active project is managed and it has one or more outside files. An "outside file" is a file that is open under the project's tab, but is not actually part of the project. When a project is closed and reopened, outside files are not remembered. Use the Close Outside Files to remove all such files from the project immediately.

Files that are removed from a project are added to the File|Open and Project|Add to Project submenus that list recently closed files, unless the files were opened in bulk withProject|Open Folder or a similar command.

## Project|Close All Files

Use the Close All command in the Project menu to close all files in the active project. The result is the same as if you had used File|Close for each file in the project. The project itself remains open.

If the project is unmanaged, the files are removed from the project.

If the project is managed, files that are part of managed projects remain part of the project as closed files. The Files Panel and Search Panel can show and search through files that are closed but are still part of managed projects. Any outside files that were open under the project's tab are removed from the project, because they were never an actual part of the project.

## Project|Reload Files from Disk

Select Reload Files from Disk from the Project menu to revert all open files in the active project to the status they had when they were last saved. This can be useful if you are viewing files that are being written to other programs.

If one or more of the files in the project have unsaved changes in EditPad Pro, or if you turned on the option to always prompt in the Open Files Preferences, then EditPad Pro will ask you to confirm which files you want to reload. Files that have unsaved changes in EditPad Pro will be marked as "Modified" in the list. Those changes will be lost if you reload those files. Tick the checkbox next to each file that you want to reload.

## Project | Save Copy of Project As

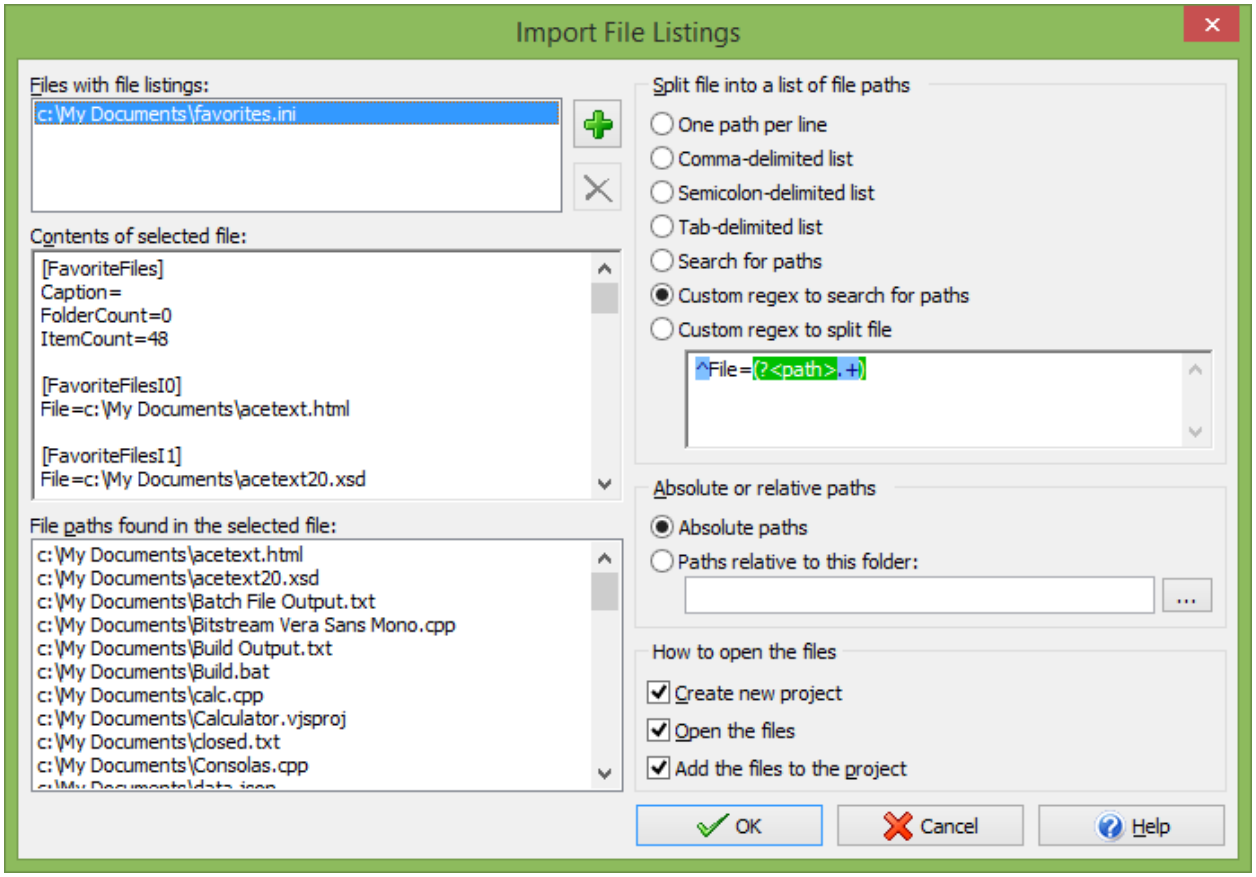
Select Save Copy of Project As to save the project under a new name, but continue updating the project under its original name. Use this command if you want to save a milestone copy of the project that you can go back to later. EditPad Pro will not keep the copy of the project updated along with the original. The new copy will stay as it is, until you open it or overwrite it when saving another project. By opening the copy of the project, you can go back to the project in the state it was in when you saved the copy.

## Project | Rename and Move Project

Select Rename/Move Project in the Project menu to save the project under a new name, and delete the original project, effectively moving the project to a new location on disk.

## Project | Import File Listing

Select Import File Listing in the Project menu to read a list of file paths from one or more text files, and then open the files indicated by the paths in that list.




In the window that appears, click the button with the green plus symbol next to the "files with file listings" list to select one or more text files that list the files or folders you want to search through. You can use the

green plus button repeatedly to add files from different folders to the list. The red X button deletes the selected file from the list.

Click on one of the files that you added to the "files with file listings" list. EditPad Pro then shows the raw contents of that file in the "contents of the selected file" box. EditPad Pro also shows the file and folder paths that has detected in that file in the "file paths found in the selected file" list. The settings on the right hand side of the Import File Listings screen determine how EditPad Pro detects those paths. EditPad Pro automatically filters out anything that does not look like a valid path.

Choose one of the options in the "split file into a list of file paths" box to tell EditPad Pro how your list of paths is delimited. If your list doesn't use a consistent delimiter, select "search for paths" to tell EditPad Pro to extract all absolute paths from the file, regardless of any other text that may occur in the file. If you want EditPad Pro to extract only certain paths from the file, select one of the two "custom regex" options and type in a regular expression in the box below them. The "custom regex to search for paths" option needs a regular expression that matches the paths you want to mark in the file selection. EditPad Pro uses the whole regex match as the path unless it contains a named capturing group called "path". E.g. «`^File=(?'path'.*)`» extracts the paths from "File" values in an .ini file. The "custom regex to split file" option needs a regular expression that matches the delimiters between those paths. E.g. «`[\r\n;]+`» allows line breaks and semicolons as delimiters.

If you select "absolute paths", EditPad Pro only uses fully qualified paths such as `c:\folder\file.txt` and `\\server\share\folder\file.txt`. Any relative paths in the file listing you're importing are ignored. If you want EditPad Pro to process relative paths as well then you need to select the "paths relative to this folder option". Type in the base folder below that option, or click the (...) button to select it from a folder tree. Note that if your file contains text in addition to paths, you need to use one of the "custom regex" options to tell EditPad Pro how to find only the actual paths. Otherwise, EditPad Pro will treat each word in the text as a file name. You cannot use the "search for paths" option because that option finds absolute paths only, regardless of the "absolute or relative paths" setting.

Once you've set the options that make EditPad Pro find the paths that you want to import, you need to tell EditPad Pro what you want to do with the files those paths point to. Tick "create new project" to open the files into a new, untitled project. This is the same as using Project|New Project before using Project|Import File Listings.

Tick "open the files" to make EditPad Pro actually open all of the files. If you turn this off, then "add files to the project" is automatically turned on, and EditPad Pro adds the files to the project without opening them. The files will show up in the Files Panel as closed files. Since only managed projects can contain closed files, turning off "open the files" automatically changes the current or the new project that the files are added into a managed project.

Tick "add the files to the project" to add the files to the project. If you turn this off, then "open the files" is automatically turned on, and the files will be opened as outside files, without adding them to the project. Since only managed projects can have files open that aren't part of the project, turning off "add the files to the project" automatically changes the current or the new project that the files are opened into a managed project.

## Recently Imported File Listings

The Import File Listings item has a submenu that lists recently imported file listings. Each time you use Project|Imported File Listing, the files you imported are added to top of this submenu. If you imported a single file, the submenu lists the full path to the file. If you imported multiple files at once, the submenu has one item with a comma-delimited list listing all the file names that fit within the maximum length for a menu item. The submenu item remembers all the settings you made in the Import File Listings window. When you select one of the remembered file listings from the submenu, it is imported again using the exact same settings.

"Remove Obsolete" removes all files that no longer exist from the Import File Listings submenu. This removes all file listings that try to import from a file that no longer exists. If a file listing imports from files that still exist, but no paths can be found in those files, then the file listing is removed as well.

## Project|Export File Listing

Select Export File Listing to save a text file that lists the full paths to all the files in the active project. The paths are delimited by line breaks. If the project is managed then open files, closed files, and outside files are all added to the exported file listing.

## Project|Delete Project

Select Delete Project in the Project menu to close the active project and delete the `.epp` file that you saved with Project|Save Project As. Only the `.epp` project file is deleted. The files that were open in the project are not deleted.

## Project|Close Project

Closes the current project and all the files in the project. If any files in the project have unsaved modifications, depending on the choice you made in Open Files Preferences, EditPad will ask if you want to save or not, automatically save, or automatically discard the changes. When you close the last project, EditPad automatically starts with a new untitled project, with one blank untitled file.

If you previously saved the project with Project|Save Project As, then the closed project is added to the top of the Project|Open Project submenu. You can use that submenu to quickly reopen the project. When you open the project again, it will show up with the same set of open files as you had open when you closed the project. The only exception are outside files in managed projects. Since outside files aren't part of the project, they aren't remembered by it and aren't reopened when you reopen the project.

If you did not previously save the project, it will be closed without asking you whether you want to save it. If you want to work with the same set of files again, you'll have to reopen the files separately.

## Project|Close All but Current

Closes all projects and all the files in them, except for the current project, as if you used Project|Close Project on each of those projects.

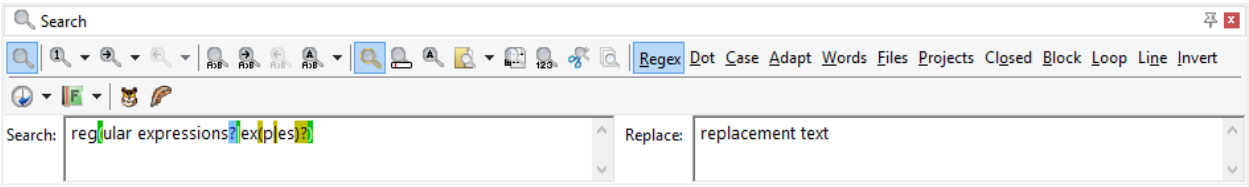
# 4. Search Menu

## Search | Prepare to Search

Use the Prepare to Search command in the Search menu to enter a search term that you can then use for searching or for searching and replacing. Its default keyboard shortcut is Ctrl+F. Exactly what the Prepare to Search command does depends on the state EditPad is in and on your preferences.

EditPad provides two search interfaces. When you start EditPad the first time, the **search toolbar** is docked to the bottom of EditPad's window. The search toolbar provides all of EditPad's search and replace commands. It has two combo boxes for entering a short single-line search term and replacement text. You can show or hide the search toolbar by right-clicking on any toolbar or the main menu and selecting Search.

If you use the Multi-Line Search Panel command, EditPad switches to the full **search panel**. This panel has two large multi-line edit controls for entering the search term and the replacement text. It also incorporates the search toolbar with all the search and replace commands, but without the two drop-down lists.




If the full search panel is visible, the Prepare to Search item puts keyboard focus in the edit box for the search term. If the search panel is not visible but the search toolbar is visible, Prepare to Search puts keyboard focus in the combo box for the search term. If the search panel and search toolbar are invisible, Prepare to Search opens the full search panel and puts keyboard focus in the edit box for the search term.

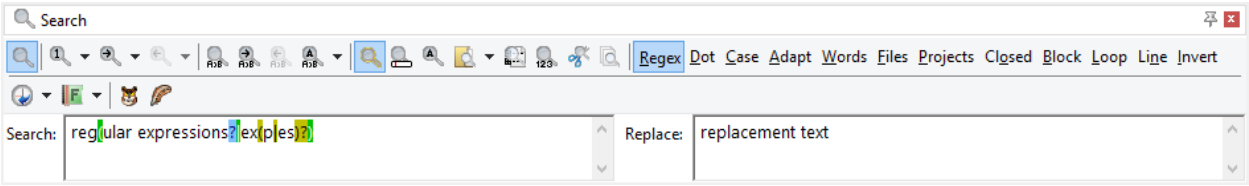
If the "use the current word or the selected text as the default search text" preference is turned on, then the Prepare to Search command uses the selected text as the default search term, as long as the selection does not span multiple lines. If there is no selection, then the word under the cursor is used as the default search term. If there is no selection and no word under the cursor, then the last used search term stays in place. If the preference is turned on, the last used search term always stays in place. The Prepare to Search command always selects the search term. You can replace the search term directly by typing in a new one.

If the "automatically turn on "selection only" when multiple lines of text are selected" preference is turned on, then the Prepare to Search command automatically turns on the Selection Only or Block search option when there is a selection that spans multiple lines. When the preference is off, the state of this option is unchanged.

## Search | Multi-Line Search Panel

EditPad provides two search interfaces. When used by itself, the **search toolbar** is docked to the bottom of EditPad's window. The search toolbar provides all of EditPad's search and replace commands. It has two combo boxes for entering a short single-line search term and replacement text.

The full **search panel** has two large multi-line edit controls for entering the search term and the replacement text. It also incorporates the search toolbar with all the search and replace commands, but without the two drop-down lists.




When you start EditPad for the first time, only the search toolbar is visible. You can switch to using the full search panel by selecting Multi-Line Search Panel in the Search menu. You can switch back to using the search toolbar only by using Multi-Line Search Panel again.

When the search toolbar is visible but the full search panel is not, you can hide the search toolbar by right-clicking on any toolbar or the main menu and selecting Search. When both the search toolbar and search panel are invisible, the Multi-Line Search Panel makes both visible as usual, but using Multi-Line Search Panel again will hide both the search toolbar and search panel again. If you want to bring back the search toolbar without the full search panel, right-click again on any toolbar or the main menu and select Search again.

To summarize: EditPad has 3 modes for its search interface: toolbar and panel hidden, toolbar visible alone, panel with toolbar visible. The Multi-Line Search Panel command switches from one of the first 2 modes to the "panel with toolbar visible" mode, and back to whichever of the first 2 modes you were using. Right-clicking on a toolbar and selecting Search switches between the first two modes.

Both the search and replace edit boxes on the search panel are full-blown multi-line edit controls, just like EditPad's main editor. They provide syntax coloring for regular expressions and match placeholders. Press Shift+Enter on the keyboard to type in and search for or replace with a newline. To type in a tab, press Shift+Tab. To switch from search to replace and back, press Tab.

If you right-click on the search or replace edit box, you will see a list of the last 16 search or replace texts you used. Click on one to use it again.

## Search | Find First

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Find First button on the search toolbar or select the Find First item in the Search menu to find the first search match. If the "selection only" option is on, Find First starts searching from the start of the search range. If "all files" is on, Find First starts searching from the start of the first file in the current project. If "all projects" is on, Find

First starts searching from the start of the first file in the first project. If none of these three options are on, Find First starts searching from the start of the current file.

If the search term can be found, EditPad will switch to the file it was found in, and select the matched text. If the search term cannot be found, nothing will happen, except that the Find First button will briefly flash its icon to indicate it failed.

In EditPad Pro, the Find First button on the search toolbar has a drop-down menu. This drop-down menu has three items that you can use to find a specific search match, counting search matches from the start. The Find 10th item will find the 10th search match. It start searching from the beginning to find the first search match, and immediately continues to find the second, third etc. until it arrives at the 10th match. The 10th match will then become selected. If there are less than 10 search matches, nothing will happen, and the Find First button will briefly flash its icon to indicate the search failed. Similarly, the Find 100th item will skip the first 99 search matches and select match #100.

The Find Nth item will ask you for the number of the match you want to find. If you enter 42, EditPad will skip the first 41 search matches, and highlight the 42nd.

## Search|Find Next

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Next button on the search toolbar or select the Find Next item in the Search menu to find the next search match. If the edit box for entering the search term still has keyboard focus, you can also find the next search match by pressing Ctrl+F again or by pressing Enter.

Find Next starts searching from the current position of the text cursor in the current file. If the "selection only" option is on, Find Next searches until the end of the search range. If "all files" is on, Find Next searches until the end of the file and then continues at the start of the next file until a search match is found. If "all projects" is on, and Find Next reaches the last file in the current project, Find Next will continue with the next project.

If the search term can be found, EditPad will switch to the file it was found in, and select the matched text. If the search term cannot be found, nothing will happen, except that the Next button will briefly flash its icon to indicate it failed.

In EditPad Pro, the Next button on the search toolbar has a drop-down menu. The first item in the drop-down menu is Find in Next File. When you select this item, EditPad will start searching from the start of the next file. You can use Find in Next File even when the "all files" option is off. If the search term cannot be found in the next file, and one or more of the "all files", "all projects" and "loop automatically" options are on, EditPad will continue the search just like the Find Next command itself.

The drop-down menu also has three items that you can use to find a specific search match, counting search matches from the start. The Find 10th Next item will skip over the next 9 search matches and highlight the 10th. The result is the same as if you had clicked the Next button 10 times. If there are no further 10 search matches, the result is slightly different. Find Next 10th will then not do anything, except flash the Next button to indicate failure. Similarly, the Find 100th Next item will skip the next 99 search matches and select match #100.

The Find Nth Next item will ask you for the number of the match you want to find. If you enter 42, EditPad will skip the next 41 search matches, and highlight the 42nd.

## Search|Find Previous

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Previous button on the search toolbar or select the Find Previous item in the Search menu to find the previous search match. Find Previous starts searching from the current position of the text cursor in the current file. If the "selection only" option is on, Find Previous searches until the start of the search range. If "all files" is on, Find Previous searches until the start of the file and then continues at the end of the previous file until a search match is found. If "all projects" is on, and Find Previous reaches the first file in the current project, Find Previous will continue with the previous project.

If the search term can be found, EditPad will switch to the file it was found in, and select the matched text. If the search term cannot be found, nothing will happen, except that the Previous button will briefly flash its icon to indicate it failed.

The Find Previous command is not available in regular expression mode. Regular expressions cannot search backwards.

In EditPad Pro, the Previous button on the search toolbar has a drop-down menu. The first item in the drop-down menu is Find in Previous File. When you select this item, EditPad will start searching from the end of the previous file. You can use Find in Previous File even when the "all files" option is off. If the search term cannot be found in the previous file, and one or more of the "all files", "all projects" and "loop automatically" options are on, EditPad will continue the search just like the Find Previous command itself.

The drop-down menu also has three items that you can use to find a specific search match, counting search matches from the start. The Find 10th Previous item will skip over the next 9 search matches and highlight the 10th. The result is the same as if you had clicked the Previous button 10 times. If there are no 10 preceding search matches, the result is slightly different. Find 10th Previous will then not do anything, except flash the Previous button to indicate failure. Similarly, the Find 100th Previous item will skip the previous 99 search matches and select match #100.

The Find Previous Nth item will ask you for the number of the match you want to find. If you enter 42, EditPad will skip the 41 preceding search matches, and highlight the 42nd.

## Search|Find Last

Select the Find Last item in the Search menu to find the last search match. If the "selection only" option is on, Find Last starts searching from the end of the search range. If "all files" is on, Find Last starts searching from the end of the last file in the current project. If "all projects" is on, Find Last starts searching from the end of the last file in the first project. If none of these three options are on, Find Last starts searching from the end of the current file. Find Last searches backwards until a match is found.

If the search term can be found, EditPad will switch to the file it was found in, and select the matched text. If the search term cannot be found, nothing will happen, except that the First button on the search panel will briefly flash its icon to indicate it failed. Since there's no Last button, the First button flashes instead.

In EditPad Pro, the Find Last command is also available through the drop-down menu of the First button on the search panel's toolbar.

The Find Last command is also available in regular expression mode, even though regular expressions cannot search backwards. Instead of starting at the end of the file, EditPad Pro starts searching at the start of the file and searches forward, skipping over all regex matches until it finds the last one. This means the Find Last command will be noticeably slower on large files when used with a regular expression rather than a plain search term.

## Search | Replace Current

After finding a search match with Find First, Find Next, Find Previous, or any of the other search commands, click the Replace button on the search toolbar or select the Replace Current item in the Search menu to replace the selected search match with the replacement text.

## Search | Replace and Find Next

After finding a search match, click the Next button to the right of the Replace button on the search toolbar or select the Replace and Find Next item in the Search menu to replace the selected search match with the replacement text, and to immediately continue searching for the next search match. This button is a shortcut to clicking the Replace button followed by clicking the Find Next button.

## Search | Replace and Find Previous

After finding a search match, click the Previous button to the right of the Replace button on the search toolbar or select the Replace and Find Previous item in the Search menu to replace the selected search match with the replacement text, and to immediately continue searching for the previous search match. This button is a shortcut to clicking the Replace button followed by clicking the Find Previous button.

The Replace and Find Previous command is not available in regular expression mode. Regular expressions cannot search backwards.

## Search | Replace All

After using Prepare to Search or pressing Ctrl+F and entering a search term and replacement text, click the Replace All button on the search toolbar or select the Replace All item in the Search menu to find the first search match. Replace All starts searching from the beginning just like Find First does. If it finds a match, it replaces it like Replace Current would, and then continues searching like Find Next would, replacing all further search matches.

The only difference between clicking Replace All and using all the individual action buttons is that Replace All is silent. It will simply replace all matches without moving the selection or the text cursor. Even when searching through all projects or all files, the current file will remain active throughout the operation.

If no search matches can be found, EditPad will briefly flash the Replace All button's icon to indicate failure. If you used the keyboard shortcut to do the Replace All with the search panel hidden, check the status bar for the result.

If there are a lot of search matches, replacing them all might take a while. The status bar will indicate EditPad's progress.

In EditPad Pro, the Replace All button on the search toolbar has a drop-down menu with two items: Replace All Next and Replace All Previous. Replace All Next performs the search-and-replace on the part of file after the position of the text cursor. If a search match is already highlighted, that match is replaced too. Effectively, Replace All Next does in one click what you can do by clicking Replace and Find Next until no further matches can be found.

The Replace All Previous button does the same, except that is searches backwards, performing the search-and-replace on the part of the file before the position of the text cursor. If a search match is already highlighted, that match is replaced too. Replace All Previous is not available in regular expression mode. Regular expressions cannot search backwards.

## Search|Highlight All

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Highlight button on the search toolbar or select the Highlight All item in the Search menu to turn on the search match highlighting mode. In this mode, all search matches will be highlighted. The Selection Only option is ignored. You can configure the "highlighted search match" color in Options|Configure File Types|Colors.

When you switch between files, search matches in the newly activated line will be highlighted, regardless of the "all files" setting. To stop highlighting search matches, simply click the Highlight button again.

## Search|Instant Highlight

The Instant Find Next command doesn't have a button on the search toolbar because it doesn't use the search term you've entered on the search panel or any of the search options. It is listed in the Search menu, primarily so you can assign it a keyboard shortcut in the Keyboard Preferences. The default shortcut is Ctrl+Alt+H, which is the most instant way of doing an Instant Highlight. If you turn on the "double-clicking a word instantly highlights all occurrences of that word" search preference, then double-clicking a word also turns on Instant Highlight.

Select Instant Highlight in the Search menu to highlight all occurrences of the selected text or the word under the cursor, without using the search toolbar or panel. Occurrences of the word as part of longer words are also highlighted. Differences in case are ignored. Highlighting persists when you switch between files. All search options, including Case Sensitive, Whole Words Only, and All Files, are ignored.

Use Instant Highlight with the same selection or on the same word to turn it off. Only one search term can be highlighted at any time. If you use both Highlight All and Instant Highlight, using either command turns off the other.

## Search|Incremental Search

Click the Incremental button on the search toolbar or select the Incremental item in the Search menu to toggle incremental search mode on or off. When off, entering a search term has no immediate effect. When on, editing the search term will cause EditPad Pro to immediately start looking for the next search match, starting from the current position of the text cursor.

While you edit the search term in incremental search mode, EditPad Pro remembers previous search matches as you type. If you type "Test", EditPad Pro will search for "T" starting from the position of the text cursor. As you type, it will search for "Te" beginning at the start of the match for "T", then for "Tes" at the start of the match for "Te", and finally for "Test". If there is no search match, nothing happens.

When you press Backspace, EditPad Pro will go back to the previously highlighted match for "Tes". If you continue pressing Backspace, it will go back to the match for "Te", "T", and finally go back to where the search started. So while typing in the search term, you can easily change your mind about the search term you're looking for by backing up with the Backspace key, and typing in the new search term. Backing up does not cause EditPad Pro to forget match positions. If you type in "Joe" after erasing "Test", EditPad Pro will search for "J" until "Joe" beginning from the original starting position. If you then back up again and retype "Test", the match previously found for "Test" is highlighted.

What does cause EditPad Pro to forget previous incremental search matches is taking any action in the main editor, including something as trivial as moving the text cursor or switching to another file. EditPad Pro will then move the starting position for the next incremental search to the new position of the text cursor.

The incremental search mode respects all search options, including "all files", "selection only", "loop automatically", etc.

## Search|Instant Find Next

The Instant Find Next command doesn't have a button on the search toolbar because it doesn't use the search term you've entered on the search panel or any of the search options. It is listed in the Search menu, primarily so you can assign it a keyboard shortcut in the Keyboard Preferences. The default shortcut is Ctrl+Alt+ArrowDown, which is the most instant way of doing an Instant Find Next.

Instant Find Next looks for the next occurrence in the current file of the word under the text cursor. When the word is found, Instant Find Next selects it. If there is no word under the text cursor, or it doesn't occur again in the file, nothing happens at all. If some text is selected when you invoke Instant Find Next, and the selection does not span more than one line, Instant Find Next will search for the selected text rather than the word under the cursor.

The purpose of Instant Find Next and Search|Instant Find Previous is to quickly "walk" through the file, stopping at each spot where the word occurs. Since a successful match selects the word that was searched for, you can instantly search for further matches.

## Search | Instant Find Previous

Instant Find Previous works just like Instant Find Next, except that it searches backwards through the current file, starting at the current position of the text cursor, or the start of the selection (if any). If you press Instant Find Previous as many times as you pressed Instant Find Next, or vice versa, you'll be back where you started.

## Search | List All Matches

After using Prepare to Search or pressing Ctrl+F and entering a search term, use the List All Matches command in the Search menu to display a list of all search matches with one line of context in a side panel. All matches in the current file, current project, or all files will be listed depending on the All Files, All Projects, and Closed Files search options.

In the side panel, you can double-click a highlighted search match to activate the file it was found in and select the match. If you edit files after listing all matches, the stored match positions are adjusted so that double-clicking a search match highlights it, even if its position in the file has shifted. The actual search matches and their context are not updated when you edit files. Use the List All Matches command again to repeat the search.

## Search | Find on Disk

After using Prepare to Search or pressing Ctrl+F and entering a search term, use the Find on Disk command in the Search menu to search through the files in a particular folder, even if you don't have those files open in EditPad Pro. The disk-based search uses the search term and search options from the search toolbar or panel. Only the All Files, All Projects, and Closed Files search options are ignored.

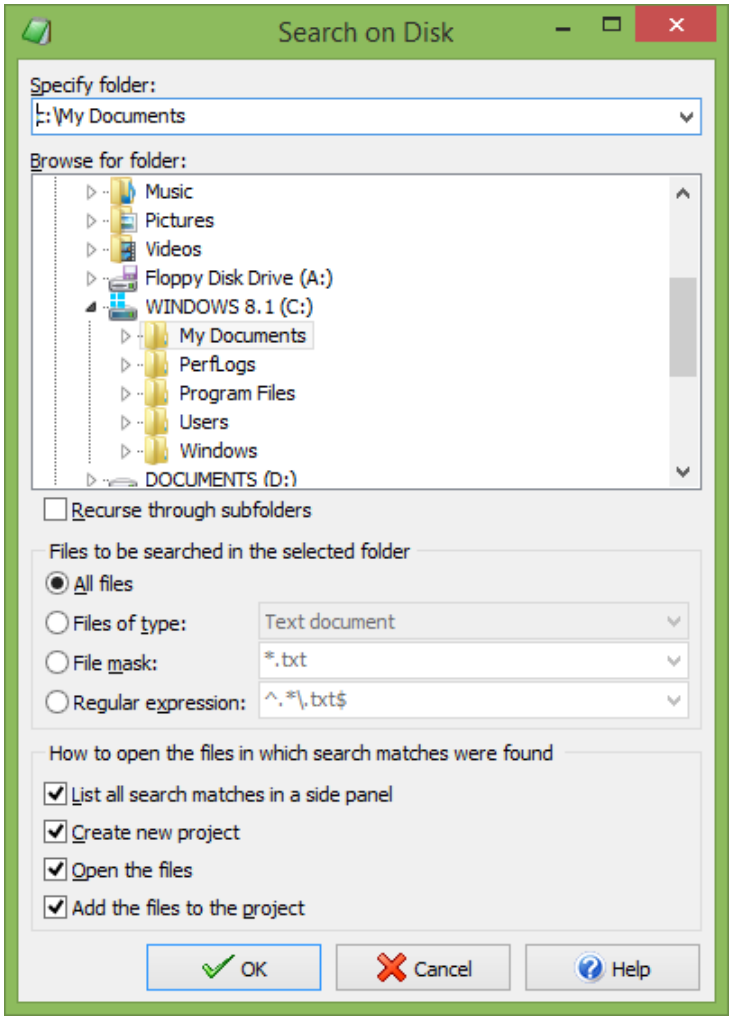
If the disk-based search includes files that you have open in EditPad Pro, then the Find on Disk command searches through the files open in EditPad Pro rather than through the files on disk. This means that if the files have unsaved changes in EditPad Pro, the unsaved changes are searched through rather than the original file on disk.

The Find on Disk command shows a folder selection screen that is very similar to the one shown by Project | Open Folder.

You can type in the path to the folder containing the files you want to search through into the drop-down list at the top. If you have recently searched through files from this folder, you can select it from the drop-down list. EditPad Pro remembers the last 16 folders. Another way to select the folder is to browse for it using the folder tree. The drop-down list and folder tree are automatically kept in sync.

Mark the option "recurse through subfolders" if you also want to search through files contained in subfolders of the folder you selected.

If you want to search through all files in the folder (and optionally its subfolders), select the "all files" option. If you want to search through only files from a certain type, you can select "files of type" and pick one of the file types from the drop-down list. The list will contain all the file types you defined in Configure File Types. This will search through all files that match the file type's file mask.




To search through files matching another list of extensions, select "file mask" and type in the list of extensions or file masks, separated by semicolons, into the "file mask" drop-down list. You can also select a recently used file mask from the drop-down list.

If you choose the "regular expression" option, then the regular expression must find a (partial) match in the name of a file for it to be included in the search. This regex is used to filter files by their names only. The text or regular expression that you want to use to search through the contents of the files needs to be specified on the search panel or in the search toolbar before you use the Find on Disk command.

Tick "list all matches in a side panel" to list all the search matches in all the files with one line of context in a side panel. This option makes Find on Disk work like Search|List All Matches, except that it searches through files on disk rather than files open in EditPad.

Tick "create new project" to open files with search matches into a new, untitled project. This is the same as using Project|New Project before using Search|Find on Disk. This option has no effect if "open the files" and "add the files to the project" are both turned off.

Tick "open the files" to make EditPad Pro actually open the files in which search matches are found. If you turn this off, but turn on "add the files to the project", then the files will show up in the Files Panel as closed files. Since only managed projects can contain closed files, turning off "open the files" automatically changes the current or the new project that the files are added into a managed project.

Tick "add the files to the project" to add the files in which search matches are found to the project. If you turn this off, but turn on "open the files", then the files will be opened as outside files, without adding them to the project. Since only managed projects can have files open that aren't part of the project, turning off "add the files to the project" automatically changes the current or the new project that the files are opened into a managed project.

## Recently Searched Through Folders

The Find on Disk item has a submenu that lists recently searched through folders. Each time you use Search|Find on Disk, the folder you search through is added to top of this submenu. Though the menu

shows only the folder's path, all the settings from the Find on Disk window are remembered by the menu. Select Maintain List in the submenu to see the settings used for each folder. When you select one of the remembered folders, EditPad Pro immediately searches though that folder, using the same settings for displaying search matches and opening files. If you have searched through the same folder more than once, the folder will appear only once in the Reopen Folder menu, with the most recently used settings.

The Find on Disk submenu does not remember the search term and other options set on the search toolbar or search panel. When you select a folder from the Find on Disk submenu, the present search term and search options are used. This way you can easily run different searches through the same folder.

"Remove Obsolete Folders" removes all folders that no longer exist from the Open Folders submenu. "Remove All Folders" clears the Find on Disk submenu.

## Search|Fold Lines

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Fold button on the search toolbar or select the Fold Lines item in the Search menu to fold all lines in which the search term cannot be found underneath lines in which it can be found. The result is that only lines in which the search term can be found will remain visible. The exception is the very first line in the file, which will remain visible even if it doesn't have any search matches. All lines until the first line with a search match will be folded underneath the very first line in the file.

If there are no search matches at all, EditPad will briefly flash the Count button's icon to indicate failure. If there are a lot of search matches, folding them all might take a while. The status bar will indicate EditPad's progress.

Using Fold Lines together with Highlight All is a very quick way to get a condensed view of all the matches in the file. Unlike the match highlighting mode, Fold Lines is automatically disabled when you switch between files, or edit the search term. Click the Fold button again to refold the current file. Folding a file based on search matches deletes all automatic folding points, and all folding points that you created. When you disable the search match folding, new automatic folding points will appear. Also unlike match highlighting, which instantly updates itself, the folding is not updated as you insert or delete the search term while you edit the file.

After folding the lines with search matches, you can use Fold|Copy Visible Lines to copy the lines with search matches to the clipboard, and Fold|Delete Folded Lines to delete the lines without search matches. However, since folding keeps the first line visible, the very first line in the file will be copied or remain behind, even if it doesn't have any search matches.

## Search|Count Matches

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Count button on the search toolbar or select the Count Matches item in the Search menu to count all search matches. Count Matches starts searching from the beginning just like Find First does. Then it continues searching like Find Next would, until the search fails. None of the search matches will be selected. After the search, Count Matches will tell you how many matches it found.

If there are no search matches at all, EditPad will briefly flash the Count button's icon to indicate failure. If there are a lot of search matches, counting them all might take a while. The status bar will indicate EditPad's progress.

## Search|Cut Matches

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Cut button on the search toolbar or select the Cut Matches item in the Search menu to cut all search matches to the clipboard. All search matches are placed onto the clipboard delimited by line breaks. The search matches are then removed from the files in EditPad.

## Search|Copy Matches

After using Prepare to Search or pressing Ctrl+F and entering a search term, click the Copy button on the search toolbar or select the Copy Matches item in the Search menu to copy all search matches to the clipboard. All search matches are placed onto the clipboard delimited by line breaks.

## Search Options

EditPad offers 12 search options. By default all these have abbreviated buttons on the Search toolbar. You can toggle them with Alt+letter key combinations when the search toolbar or search panel has keyboard focus.

Regular Expression (Regex)
Dot Matches Newline (Dot)
Case Sensitive (Case)
Adapt Case (Adapt)
Whole Words Only (Words)
All Files (Files)
All Projects (Projects)
Closed Files (Closed)
Selection Only (Block)
Loop Automatically (Loop)
Line by Line (Line)
Inverted Line by Line (Invert)

## Search Option: Regular Expression (Regex)

EditPad's search-and-replace module includes a full-featured regular expression engine. Its regular expression flavor is compatible with popular regular expression flavors such as those used by Perl, Java, and .NET. In the regular expression flavor comparison, EditPad's flavor is indicated as "JGsoft". To search using a regular expression, simply turn on the regular expression search option by clicking the Regex button on the search toolbar.

When using capturing groups in a regular expression, you can insert the groups' contents in the replacement text using backreferences `\0`, `\1`, `\2`, etc. `\0` inserts the whole regex match, while `\1` inserts the text matched by the first capturing group, `\2` the second group, etc. EditPad Pro supports up to 99 backreferences. If you want to insert a backreference followed by a literal number into the replacement text, use two digits for the backreference. The replacement text `\15` will replace the regex match with the contents of the first backreference followed by the digit 5. You can also use dollar signs instead of backslashes. There's no difference between `\1` and `$1` in the replacement text in EditPad.

If you want to pad or control the case of backreferences, use the `%GROUP1%` syntax for match placeholders.

If you use «`\r`» and «`\n`» in your regex, remember that Windows text files use «`\r\n`» terminate lines. EditPad's regular expression engine matches «`\r`» and «`\n`» to a single CR and a single LF. If you want differences in line breaks to be handled automatically, use the full search panel and press Shift+Enter to type in an actual line break into your regular expression. EditPad's regular expression engine will match each literal line break in your regular expression to either a CRLF pair, or a single LF, or a single CR.

The caret and dollar always match at the start and the end of a line, in addition to the start and the end of a file. EditPad Pro does not have an option to change this. To match the start or the end of the file only, use \A or \z instead. The behavior of the dot can be changed with the "dot matches newline" option.

EditPad Pro does not have a search option to use free-spacing regular expressions. It does support the mode modifier «`(?x)`». If you put «`(?x)`» at the very start of your regular expression, then any whitespace after the «`(?x)`» is ignored, unless it is escaped or inside a character class.

## Search Option: Dot Matches Newline (Dot)

When searching for a regular expression, you can click the Dot button on the search toolbar to turn on the "dot matches newline" search option. This option makes the dot match any character, including line break characters.

Turning on this option is the same as putting the mode modifier «`(?s)`» at the start of your regular expression. If you do use the mode modifier «`(?s)`» or «`(?-s)`» then the mode modifier overrides the state of the checkbox.

## Search Option: Case Sensitive (Case)

If you click the Case button on the search toolbar to turn on the "case sensitive" search option, then the difference between lowercase and uppercase letters is taken into account. E.g. "Dog" will considered to be different from "dog". Otherwise, "dog", "Dog", "DOG", and even "doG" are all the same.

## Search Option: Adapt Case (Adapt)

When the "case sensitive" option is off, you can click the Adapt button on the search toolbar to turn on the "adapt case" search option. When you do so, EditPad Pro will adapt the casing of the replacement text to the casing of the search match to be replaced. EditPad Pro recognizes four casings: all uppercase, all lowercase,

first letter capitalized, and first letter of each word capitalized. If EditPad Pro doesn't recognize the casing, the replacement text is used exactly as you entered it.

E.g. when searching for "My DoG" and replacing with "My CaT", then "my dog" will be replaced with "my cat" (all lowercase), "MY DOG" with "MY CAT" (all uppercase), "My dog" with "My cat" (first letter capitalized) and "My Dog" with "My Cat" (first letter of each word capitalized). If EditPad Pro doesn't recognize the casing, e.g. "my DOG", then "My CaT" is substituted the way you entered it.

## Search Option: Whole Words Only (Words)

Click the Words button on the search toolbar to turn on the "whole words only" search option if only entire words should be valid search results. If this option is on, when looking for "cat", EditPad will not consider the first three letters of "category" a valid search match. Otherwise, it will.

## Search Option: All Files (Files)

Click the Dot button on the search toolbar to turn on the "all files" search option to search through all open files in the current project. If a search command cannot find a match in the current file, it will automatically continue with the next file until either a match is found, or all files have been searched through.

The Fold Lines search command always works on the current file only, regardless of the "all files" option. The Highlight All option is a global toggle that will always highlight the current file, even when you switch files, until you turn it off, regardless of the "all files" option. All the other search commands respect the "all files" option as explained above.

## Search Option: All Projects (Projects)

To search through all open files in all open projects, click the Projects button on the search toolbar to turn on the "all projects" search option. If a search command cannot find a match in the current file, it will automatically continue with the next file. When all files in the current project have been searched through, EditPad Pro will continue with the first file in the next project. EditPad Pro will continue searching until either a match is found, or all files in all projects have been searched through.

The Fold Lines search command always works on the current file only, regardless of the "all projects" option. The Highlight All option is a global toggle that will always highlight the current file, even when you switch files, until you turn it off, regardless of the "all projects" option. All the other search commands respect the "all projects" option as explained above.

## Search Option: Selection Only (Block)

Click the Block button on the search toolbar to turn on the "selection only" search option to search only through the selected part of the current file.

When you invoke a search command like Find First with this option on, the selection will be turned into the search range. The search range is highlighted in a different color than the selection. You can configure both the "selected text" and "search range" colors in the Colors Preferences.

If a search match is found, that search match will become selected. You can then edit the search match or any part of the file inside or outside the search range. As long as you don't turn off the "selection only" search option, the search range will remain highlighted.

When you invoke the next search command, and there already is a search range, EditPad will either search through the existing search range when no text is selected, or when the selection falls entirely inside the search range. If some of the text outside the search range is selected, EditPad will use the selection as the new search range.

In practice, this mechanism is very intuitive. When you search, EditPad selects the search match, which always falls inside the search range. So you can click Search|Find Next and Search|Find Previous without paying much attention to the search range. It will stay put. When you want to search through a different part of the file, simply select it, make sure "selection only" is on, and search. You don't need to check whether there was a search range already. EditPad will search through the block of text that you selected.

## Search Option: Loop Automatically (Loop)

Click the Loop button on the search toolbar to turn on the "loop automatically" search option if you want Find Next, Find Previous, Replace and Find Next and Replace and Find Previous to automatically restart searching from the start or end of the file when no further search matches can be found. If the "all files" option is on, EditPad will restart from the first file (find next) or last file (find previous) in the project. If "all projects" is also on, EditPad will restart from the first file in the first project (find next), or the last file in the last project (find previous).

When EditPad finds a match after having looped (i.e. restarted from the start or end), then the glyph on the search command button that you clicked will flash briefly, showing a "looped" glyph.

## Search Option: Closed Files (Closed)

When the option to search through all files or all projects is turned on, and one of the project's you're searching through is a managed project that contains closed files, you can click the Closed button on the search toolbar to turn on the "closed files" option to search through the closed files in addition to searching through the open files.

Commands that select found matches such as Find Next and commands that modify files such as Replace All automatically open closed files in which matches are found so that the found match can be selected or the file can be modified. Closed files in which no search matches are found are not opened.

## Search Option: Line by Line (Line)

Click the Line button on the search toolbar to turn on the "line by line" search option to search through each line separately, and to treat the entire line as the search match if part of it matches. This means that when this

option is on, a search match can never span across multiple lines. Commands like Replace All and Copy Matches will replace or copy entire lines, even when the search term matches only part of the line.

## Search Option: Inverted Line by Line (Invert)

When using the Line by Line search option, you can turn on Inverted Line by Line to match all lines in which the search term cannot be found. Click the Invert button on the search toolbar to toggle this option.

## Search|History

Click the History button on the search panel to switch between the currently displayed search term and search options, and the ones you used before. This way you can easily work with two search actions. E.g. if you need to do a search-and-replace in several blocks in a file, and you want to find the block by searching for another word, first search for the word and then do the search-and-replace in the first block. Then click the History button to instantly get back the search term. Find the second block, and click the History button again to get the search-and-replace search term and options.

EditPad's search function remembers up to 16 search settings. You can access them via the submenu of the History button. Pick a search term from the menu to use it again. All the Search Options will also be set the way they were when you last used the search term you just picked from the history.

If you only want to recall the search text or the replacement text, without changing any of the search options, right-click on the search box or replace box. The right-click menu will list the last 16 search texts or replacement texts.

## Search|Favorites

If you regularly use a particular search term, you should add it to your search favorites. To do so, select Search|Favorites|Add Search Term in the menu. Or, click on the downward pointing arrow next to the Favorites button on the search panel, and select Add Search Term in the drop-down menu. The search text, replacement text, and all search options will be stored in the favorites. To make it easy to differentiate between search terms, you'll be asked for a label when adding a favorite search term.

If you click directly on the Favorites button rather than the arrow next to it, the Organize Favorites screen will appear. Select a search term and click on the Use button to set the search text, replacement text and search options to those stored in the favorites item. Click the Remove button to delete a favorite.

With the New Folder and Rename Folder, you can organize your favorite search terms into folders. Organizing favorites into folders makes it easier to retrieve the search term you want. To move favorites into a folder, simply drag-and-drop them with the mouse.

## Search | RegexBuddy

Use regular expressions with EditPad Pro's search and replace to automate many editing task that would be tedious to do manually. The regex syntax is quite terse, making complex regular expressions sometimes difficult to interpret.

To alleviate this complexity, we have developed a new product called RegexBuddy. RegexBuddy makes it very easy to write and edit regular expressions. Use RegexBuddy's detailed and descriptive regex tree, and the easy-to-grasp regex building blocks instead of or in combination with the regex syntax to define your search patterns.

Click the RegexBuddy button on the search panel or select the RegexBuddy item in the Search menu to start RegexBuddy. The search and replace texts you entered in EditPad Pro are automatically sent to RegexBuddy for editing. When you have prepared the new regular expression and replacement text, click the Send button in RegexBuddy. RegexBuddy will then close and send the new texts to EditPad Pro. If you change your mind, close RegexBuddy without clicking the Send button.

You can also use RegexBuddy to test your search patterns in a safe and intuitive sandbox. Very handy is the ability to collect your own libraries of regular expressions. Or use RegexBuddy's standard library of regular expressions for common search patterns.

You can use RegexBuddy in combination with EditPad Pro, as well as with any other application you use regular expressions with. If you are a programmer, RegexBuddy has full support for a variety of programming languages and regex libraries.

See http://www.regexbuddy.com for more information about RegexBuddy.

Please note that RegexBuddy is a separate product. If you want to use RegexBuddy with EditPad Pro, you will need to purchase both products.

## Search | RegexMagic

Use regular expressions with EditPad Pro's search and replace to automate many editing task that would be tedious to do manually. The regex syntax is quite terse, making complex regular expressions sometimes difficult to interpret.

To make it easy to create regular expressions, we have developed RegexMagic. Instead of dealing with the cryptic regular expression syntax, use RegexMagic's powerful patterns for matching characters, text, numbers, dates, times, email addresses, URLs, card numbers, etc. RegexMagic can detect these patterns when you mark your sample text. You can combine multiple patterns to match exactly what you want.

Click the RegexMagic button on the search panel or select the RegexMagic item in the Search menu to start RegexMagic. When you have generated the new regular expression and replacement text, click the Send button in RegexMagic. RegexMagic will then close and send the new regulare expression to EditPad Pro. If you change your mind, close RegexMagic without clicking the Send button.

Use RegexMagic in combination with EditPad Pro, as well as any other application you use regular expressions with. RegexMagic supports all popular regular expression flavors. If you are a programmer, RegexMagic has full support for a variety of programming languages and regex libraries.

See http://www.regexmagic.com for more information about RegexMagic.

Please note that RegexMagic is a separate product. If you want to use RegexMagic with EditPad Pro, you will need to purchase both products.

## Match Placeholders

Match placeholders can be used to insert search matches or match counts in the search text or replacement text.

| Placeholder | Meaning and Examples | Availability |
|---|---|---|
| %LINE% | The line the search match was found on. | Replacement text. |
| %LINEN% | The number of the line the match was found on. | Replacement text. |
| %MATCH% | The search match. | Replacement text. |
| %MATCHN% | The number of the search match. Counts the number of search matches already found. If used in the search term, %MATCHN% will be zero upon the first search attempt. If used in the replacement text, %MATCHN% will be one for the first replacement. | Search term and replacement text. |
| %GROUP1%, %GROUP2%, etc. | %GROUP1% is a backreference to a capturing group, equivalent to «`\1`» in a regular expression or "`\1`" or "`$1`" in the replacement text. The advantage of %GROUP1% is that you can use the padding and case conversion specifiers listed below. | Search term and replacement text, but only when using regular expressions with numbered capturing groups. |
| %GROUPNAME%, %GROUPanothername%, etc. | %GROUPNAME% is a backreference to a named capturing group, equivalent to «`\k'NAME'`» or «`(?P=NAME)`» in a regular expression and to "`${NAME}`" or "`\g<NAME>`" in the replacement text. The NAME part of the placeholder is case sensitive. The advantage of %GROUPNAME% is that you can use the padding and case conversion specifiers listed below. | Search term and replacement text, but only when using regular expressions with named capturing groups. |

## Padding

You can add additional specifiers to all of the above placeholders. You can pad the placeholder's value to a certain length, and control the casing of any letters in its value. The specifiers must appear before the second % sign in the placeholder, separated from the placeholder's name with a colon. E.g. %MATCH:6L% inserts the match padded with spaces at the left to a length of 6 characters. You can add both padding and case placeholders. %MATCH:U:6L% inserts the padded match converted to upper case.

Padding specifiers start with a number indicating the length, followed by a letter indicating the padding style. The length is the number of characters the placeholder should insert into the regular expression or replacement text. If the length of the placeholder's value exceeds the requested length, it will be inserted unchanged. It won't be truncated to fit the length. If the value is shorter, it will be padded according to the padding style you specified.

The L or "left" padding style puts spaces before the placeholder's value. This style is useful for padding numbers or currency values to line them up in columns. The R or "right" padding style puts spaces after the placeholder's value. This style is useful for padding words or text to line them up in columns. The C or "center" padding style puts the same number of spaces before and after the placeholder's value. If an odd number of spaces is needed for padding, one more space will be placed before the value than after it.

The Z or "zero" padding style puts zeros before the placeholder's value. This style is useful for padding sequence numbers. The A or "alpha" padding style puts letters "a" before the placeholder's value. This style is useful for padding sequence letters like %MATCHNA%.

Padding style letters are case insensitive, except for the "alpha" style. %matchna:6a% uses lowercase letters, and %MATCHNA:6A% uses uppercase letters.

## Case Conversion

Case conversion specifiers consist of a letter only. The specifier letters are case insensitive. Both "U" and "u" convert the placeholder's value to uppercase. L converts it to lowercase, I to initial caps (first letter in each word capitalized) and F to first cap only (first character in the value capitalized). E.g. %MATCH:I% inserts the match formatted as a title.

Instead of a case conversion specifier, you can use the case adaptation specifier. It consists of A followed by a digit (not to be confused with a number followed by A, which is a padding specifier). This specifier is only available in a replacement text corresponding with a regular expression. You can use the specifier to give the placeholder the same casing style as the regular expression match or the text matched by a capturing group. Specify zero for the whole regex, or the backreference number of a capturing group. E.g. %MATCH:A2% inserts the whole regular expression match, converted to the case used by the second capturing group. The case adaptation specifier detects and adapts to uppercase, lowercase, initial caps and first cap. If the referenced capturing group uses a mixed casing style, the placeholder's value is inserted unchanged.

## Arithmetic

Arithmetic specifiers perform basic arithmetic on the placeholder's value. This works with any placeholder that, at least prior to padding, represents an integer number. If the placeholder does not represent a number,

the arithmetic specifiers are ignored. For placeholders like %MATCH% that can sometimes be numeric and other times not, the arithmetic specifiers are used whenever the placeholder happens to represent an integer.

An arithmetic specifier consists of one or more operator and integer pairs. The operator can be `+`, `-`, `*`, or `/` to signify addition, subtraction, multiplication, or integer division. It must be followed by a positive integer. Arithmetic specifiers are evaluated strictly from left to right. When %MATCHN% evaluates to 2, %MATCHN:+1*2% evaluates to 6 because 2+1=3 and 3*2=6. Multiplication and division do not take precedence over addition and subtraction. Integer division drops the fractional part of the division's result, so %MATCHN:/3% evaluates to 0 for the first two matches.

# 5. Go Menu

## Go|Go to Line or Offset

When editing a text file, the Go to Line item in the Go menu will ask you for a line number, and then move the text cursor to that line. You can enter the line number as a decimal number (just enter the number itself) or as a hexadecimal number (as 0xFF or FFh).

When editing a file in hexadecimal mode, the Go to Offset command will ask you for a file offset, counting the number of bytes from the start of the file. You can enter the offset in hexadecimal, decimal, octal or binary notation. You don't need to format the number in any way (e.g. just enter FF for a hexadecimal offset). Just click the radio button with the notation you're using.

Whether going to a line or offset, you can prefix the line or offset number with a plus or minus sign. The sign makes the move relative to the current line or offset. If the cursor is at line 150, and you go to "100", the cursor is moved to line 100. If you enter "-100" it is moved to line 50, while +100 moves it to line 250. You can prefix numbers in any of the four notations with a sign. E.g. entering a hexadecimal offset "-F" moves the cursor fifteen bytes to the left.

## Go|Go to Matching Bracket

On the Colors & Syntax tab in the file type configuration, you can make EditPad Pro highlight matching brackets such as ( and ) for certain file types. When the text cursor is on a character that is considered a bracket by the syntax coloring scheme, then both that bracket and its matching counterpart are highlighted. In that situation, you can select Go to Matching Bracket from the Go menu to move the text cursor to the matching bracket.

## Go|Previous Editing Position

Select Previous Editing Position in the Go Menu to move the text cursor to the position in the current file where you last inserted or deleted some text. This makes it very comfortable to scroll through a file to look something up while editing, since you can always jump back to the editing position where you left off to continue editing.

If you use the Go to Previous Editing Position command a second time right after the first time, the cursor will be moved to the editing position before the previous one. If you use it a third time, the cursor will be moved to the editing position before that. You can use the command up to 16 times to revisit the last 16 spots in the file where you made changes.

If you go back too far, use the Go|Next Editing Position command to move forward through the most recent editing positions.

EditPad remembers the last 16 editing positions for all files that you have open. The Go to Previous Editing Position command never switches between files. You can use Go|Back in Edited Files and Go|Forward in Edited Files to switch between recently edited files.

## Go|Next Editing Position

If you have used Go|Previous Editing Position to go back to a position where you previously made a change in the active file, then you can use the Next Editing Position command in the Go menu to move forward in the list of recent editing positions.

## Go|Back in Edited Files

Select Back in Edited Files in the Go menu to activate the last file that you made a change to. If that file is already active, the file you last made a change to other than the active file will be activated. When you have many files open, you can use this command to quickly go back to the file you were editing after you've looked at other files. E.g. if you want to copy and paste something from another file into the file you're editing, switch to the other file, copy the text you want to paste, and then use Go|Back in Edited Files to instantly go back to the spot where you want to paste.

If you use the Previously Edited File command a second time, EditPad will activate the second most recently edited file. EditPad remembers up to 16 files. You can also use the submenu of the Back in Edited Files command to directly select the file you want to go back to.

Essentially, the Back in Edited Files and Forward in Edited Files commands work like the Back and Forward buttons in a web browser. But they only remember files that you've actually edited, not every file you've viewed.

## Go|Forward in Edited Files

If you have used Go|Back in Edited Files to go back to a previously edited file, then you can use the Forward in Edited Files command in the Go menu to move forward in the list of recently edited files. You can also select a particular file to go to in the submenu of the Forward in Edited Files command.

## Go|Next File

The Go|Next File menu item activates the next file tab in the current project. If the last file in the project is active, then the first file is activated.

If you record this command as part of a macro and the macro plays back the Go|Next File command when the last file is already active, then macro playback is stopped. The first file is not activated. This makes it possible to record a macro that operates on all files by recording Go|Next File as the last command in the macro. Then you can play back the macro and repeat it any number of times, knowing that it will stop at the last file.

The Next File and Previous File commands always switch between files in the order their tabs have. If you want to switch between recently edited files, use the Go|Back in Edited Files and Go|Forward in Edited Files commands. If you want to switch between recently viewed files, regardless of whether you've edited them, use the Ctrl+Tab key combination with the "use most recent order when switching tabs with Ctrl+Tab" preference turned on. If you quickly want to go to a specific file, use the Files Panel.

## Go|Previous File

The Go|Previous File menu item activates the previous file tab in the current project. If the first file in the project is active, then the last file is activated. During macro playback, if the first file is active, macro playback is stopped without switching between files. See Go|Next File for details.

## Go|Sort File Tabs Alphabetically

Pick Sort File Tabs Alphabetically from the Go menu to sort the tabs of all files in the current project alphabetically by their file names. Note that the tabs will be sorted only at the moment you click on Sort File Tabs Alphabetically. If you open more files, the alphabetical order will not be maintained. Tabs for newly opened files are either placed to the immediate right hand side of the tab that was active when you opened the additional files, or after the last file tab. You can configure this in the Tabs Preferences.

You can also rearrange the tabs in any order you want by dragging them with the mouse. To drag a tab, click on it, hold the mouse button down, and move the mouse to the left or right. The tab will follow the mouse pointer until you release the mouse button. If you drag the tab to the left edge of the leftmost tab, and there are more tabs, presently invisible, to the left of that tab, then all the tabs will scroll to the right, allowing you to move the tab you are dragging further to the left. Similarly, you can drag the tab to the right edge of the rightmost tab to make the tabs scroll to the left.

## Go|Next Project

The Go|Next Project item activates the next project tab. If the last project is active, then the first project will be activated. During macro playback, if the last project is active, macro playback is stopped without switching between projects.

If you like to use the mouse, you can quickly switch between projects by clicking on their tabs, or by double-clicking on the project's node in the Files Panel.

## Go|Previous Project

The Go|Previous Project menu item activates the previous project tab. If the first project is active, then the last project is activated. During macro playback, if the first project is active, macro playback is stopped without switching between projects.

If you like to use the mouse, you can quickly switch between projects by clicking on their tabs, or by double-clicking on the project's node in the Files Panel.

## Go|Sort Project Tabs Alphabetically

Pick Sort Project Tabs Alphabetically from the Go menu to sort the tabs of all projects alphabetically by their names. Note that the tabs will be sorted only at the moment you click on Sort Project Tabs Alphabetically. If you open more projects, the alphabetical order will not be maintained. Tabs for newly opened projects are

always placed to the immediate right hand side of the tab that was active when you opened the additional files.

You can also rearrange the tabs in any order you want by dragging them with the mouse. To drag a tab, click on it, hold the mouse button down, and move the mouse to the left or right. The tab will follow the mouse pointer until you release the mouse button. If you drag the tab to the left edge of the leftmost tab, and there are more tabs, presently invisible, to the left of that tab, then all the tabs will scroll to the right, allowing you to move the tab you are dragging further to the left. Similarly, you can drag the tab to the right edge of the rightmost tab to make the tabs scroll to the left.

# 6. Block Menu

## Block|Duplicate

Select Duplicate from the Block menu to insert a copy of the selected text at the current position of the text cursor. The newly inserted copy will become the selected text.

This command is intended to be used with persistent selections. You can move the text cursor without clearing the selection only when selections are persistent.

Selecting text, moving the text cursor, and invoking Block|Duplicate has the same effect as selecting text, invoking Edit|Copy, moving the text cursor, and invoking Edit|Paste. The only difference is that Block|Duplicate does not use the clipboard, so the contents of the clipboard are preserved.

The option "paste whole lines when lines are copied as a whole" affects how text is duplicated when you have selected complete lines. Selecting a complete line means to select everything from the start of the line to the end of the line, including the line break at the end of the line. Selecting multiple lines completely means selecting everything from the start of the first line in the block until the end of the last line in the block, including the line break at the end of the last line. When the option "paste whole lines when lines are copied as a whole" is on, whole lines are always duplicated as if the cursor were at the start of the line when you invoke Block|Duplicate. Thus, whole lines are always duplicated as a whole before the line that the cursor is on when duplicating. This makes it easy to duplicate blocks of lines without worrying about the horizontal position of the text cursor. If this option is off, the selected lines are duplicated at the exact spot the text cursor is at, even when whole lines are selected.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Block|Move

Select Move from the Block menu to delete the selected text and insert it again at the current position of the text cursor. Block|Move will only become enabled when you have enabled persistent selections in the Editor Preferences. Only when selections are persistent, you can move the text cursor without clearing the selection.

Selecting text, moving the text cursor, and invoking Block|Move has the same effect as selecting text, invoking Edit|Cut, moving the text cursor, and invoking Edit|Paste. The only difference is that Block|Move does not use the clipboard, so the contents of the clipboard are preserved.

The option "paste whole lines when lines are copied as a whole" affects how text is moved when you have selected complete lines. Selecting a complete line means to select everything from the start of the line to the end of the line, including the line break at the end of the line. Selecting multiple lines completely means selecting everything from the start of the first line in the block until the end of the last line in the block, including the line break at the end of the last line. When the option "paste whole lines when lines are copied as a whole" is on, whole lines are always moved as if the cursor were at the start of the line when you invoke Block|Move. Thus, whole lines are always moved as a whole before the line that the cursor is on when

moving. This makes it easy to move blocks of lines without worrying about the horizontal position of the text cursor. If this option is off, the selected lines are moved to the exact spot the text cursor is at, even when whole lines are selected.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Block|Swap Selections

If you have used View|Split Editor to split EditPad Pro's editor in two, you can select a different part of the active file in each of the split editors. You can then use the Swap Selections item in the Block menu to swap the two selected blocks. If your file contains the sentence "Harry met Sally" and you select "Harry" in one view and "Sally" in the other view, then Block|Swap Selections changes the sentence into "Sally met Harry".

## Block|Indent

Select Indent from the Block menu to insert a certain number of spaces at the start of each line in the selected portion of the current file. How many spaces are inserted depends on the "block indent" setting for the type of file you are editing. You can change this setting on the Editor page in the file type configuration. If the "block indent" setting is an integer multiple of the "tab size" setting, and the option "tab inserts spaces" is off, Block|Indent will use tab characters instead of spaces to indent the selection.

If the selection spans more than one line, you can also indent it by pressing Tab on the keyboard. The difference between using Block|Indent and the Tab key is that Block|Indent uses the "block indent" setting while Tab uses the "tab size" setting to determine the amount of indent. If "tab inserts spaces" is off, pressing tab will indent the selection by inserting exactly one tab character at the beginning of each paragraph.

Note that if word wrap is on, only the first line of each paragraph will be indented. Wrapped lines are only indented along with the first line if you've turned on Options|Indent Wrapped Lines.

If a rectangular block is selected, Block|Indent works differently. Instead of indenting the selected lines by inserting spaces or tabs at the start of the line, it will insert the spaces immediately before the first column in the selection, effectively shifting the block a number of spaces to the right.

## Block|Outdent

Select Outdent from the Block menu to outdent or unindent each line in the selected portion of the current file. EditPad does this by deleting a certain number of spaces from the start of each line. How many spaces are inserted depends on the "block indent" setting for the type of file you are editing. You can change this setting on the Editor page in the file type configuration. If there are not that many spaces at the beginning of a certain line, all the spaces at the beginning of that line will be deleted. But the other lines will still be outdented by the specified amount.

If the selection spans more than one line, you can also outdent it by pressing Shift+Tab on the keyboard. The difference between using Block|Outdent and the Tab key is that Block|Outdent uses the "block indent" setting while Tab uses the "tab size" setting to determine the amount.

If a rectangular block is selected, Block|Outdent works differently. Instead of deleting spaces or tabs at the start of the line, it will delete spaces and tabs immediately to the left of the first column in the selection, effectively shifting the block a number of spaces to the left. Spaces inside the selected block will not be deleted.

## Block|Move Lines and Columns

The Move Lines and Columns submenu of the Block menu lists various commands for shifting blocks of text around.

When you've made a linear selection (a selection that follows the flow of the text), you can only move lines. Lines will be moved entirely. If the first and/or last line in the selection are only partially selected, the selection will first expand itself to include those lines entirely. When moving the block one line up, the block and the line above it will swap places. After the action, the unselected line that was above the block will be below it. Moving the block one line down does the opposite.

When you've made a rectangular selection, only the selected columns are moved. When moving the rectangular block one line up, the columns spanning the width of the selection in the line above the selection are removed. Then the selected block is moved up one line, filling up the space created by removing part of the line that was above the selection. The removed characters are then reinserted in the empty space created on the line that use to be the last line in the selection. If the line above the block did not have any characters to remove, spaces are inserted into the last line instead. The original block will remain selected, now positioned one line higher in the file. Moving a rectangular block one line down does the opposite.

Rectangular selections can also be shifted to the left or the right. When shifting a block to the left, the character immediately to the left of the block on each line partially selected by the block is removed. This character is then reinserted on the same line immediately to the right of the block. The original block will remain selected, now positioned one column closer to the left. When shifting a block to the right, the character immediately to the right of the block is removed, and reinserted to the left of the block. If there is no character to the right of the block on a particular line, a space is inserted to the left of the block instead. You can shift a block as far out to the right as you want.

If no text is selected at all, you can still use the commands to move lines up or down. Then, the line containing the text cursor will be moved.

## Block|Comment

Select Comment from the Block menu to comment out a piece of code in a source code file.

If the current file is being syntax colored by a syntax coloring scheme that contains a character string for single line comments, Block|Comment will automatically comment out the using those characters. If you've made a selection that flows along with the text, the comment characters will be inserted at the very start of the line. If you've made a rectangular selection, the comment characters will be inserted on each line spanned

by the selection, at the leftmost column in the selection. This is exactly what Block|Prefix also does, except that Block|Comment automatically uses the characters to comment out something based on the syntax coloring scheme.

If the syntax coloring scheme does not define characters for single line comments, but does define characters for multi-line comments, Block|Comment will use the characters that open and close a comment on each line in the selection. If the selection is rectangular, they're inserted immediately before and after the selected columns. Otherwise, they're inserted at the start and end of each selected line.

If the current file type does not use a syntax coloring scheme, or the scheme does not define any characters to comment out something, Block|Comment will ask you for the characters to be inserted at the start of each line, just like Block|Prefix would.

If you did not select any text before using the Block|Comment command, or the selection does not span more than one line, Block|Comment will place the comment characters at the very start of the line containing the text cursor. The text cursor is then moved to the next line. If you've assigned a keyboard shortcut to Block|Comment, you can quickly comment out a bunch of lines this way.

To remove the comment characters inserted by Block|Comment, use Block|Uncomment or Block|Toggle Comment.

## Block|Uncomment

Select Uncomment from the Block menu to remove the comment characters added by Block|Comment. When you've made a selection that flows with the text, Block|Uncomment will remove the comment characters from the very start of the line only. Comment characters in the middle of a line are left. When you've made a rectangular selection, only comment characters starting at the first selected column will be removed. Comment characters at other columns are left.

If the syntax coloring scheme does not define opening and closing characters for comments, Block|Uncomment will try to remove the pair from the start and the end of each line when the selection flows with the text. When making a rectangular selection, it should include both the opening characters and closing characters on each line. Only comment characters inside and at the edge of the selected block will be removed.

Though Block|Uncomment requires you to be a bit precise when selecting the block to be uncommented, this makes it possible to comment out blocks that already have comments in them, and then uncomment those blocks while leaving the original comments.

If you did not select any text before using the Block|Uncomment command, or the selection does not span more than one line, Block|Uncomment will attempt to remove comment characters at the very start of the line containing the text cursor. Whether any comment characters were removed or not, it will then move the text cursor to the next line. If you've assigned a keyboard shortcut to Block|Uncomment, you can quickly uncomment a bunch of lines this way.

## Block|Toggle Comment

The Toggle Comment command in the Block menu combines the functionality of Block|Comment and Block|Uncomment. On each line in the selection, it will attempt to remove the comment characters exactly like Block|Uncomment would. If there are no comment characters to be removed, it will add them on that line just like Block|Comment does.

If you did not select any text before using the Block|Toggle Comment command, or the selection does not span more than one line, Block|Toggle Comment will add or remove the comment character at the start of the line containing the text cursor. The text cursor is then moved to the next line. If you've assigned a keyboard shortcut to Block|Uncomment, you can quickly uncomment a bunch of lines this way.

The advantage of Block|Toggle Comment is that it allows you to use the same keyboard shortcut for both commenting and uncommenting lines.

## Block|Prefix

Select Prefix in the Block menu to prefix the selected block with one or more characters. If you've made a selection that flows along with the text, the characters will be inserted at the very start of each line in the selection. If you've made a rectangular selection, the comment characters will be inserted on each line spanned by the selection, at the leftmost column in the selection. The inserted characters will become part of the selection.

## Block|Suffix

Select Suffix in the Block menu to suffix the selected block with one or more characters. If you've made a selection that flows along with the text, the characters will be inserted at the end of each line in the selection. If you've made a rectangular selection, the comment characters will be inserted on each line spanned by the selection, after the rightmost column in the selection. The inserted characters will become part of the selection.

## Block|Fill Columns

Select Fill Columns in the Block menu to fill a rectangular block with one or more characters. If you enter less characters than the width of the selection, the characters you entered will be repeated on each line as many times as necessary. If the number of selected columns is not an integer multiple of the number of characters you entered, the last repetition of the inserted characters will be truncated at the last selected column. The same characters will be inserted on each line.

If some of the lines already had text in the selected columns, that text is overwritten by the filled block. If some of the lines were shorter than the selected columns, spaces will be inserted to extent those lines until the filled block.

## Block|Rectangular Selections

There are several ways to select text in EditPad using the keyboard or the mouse. Whichever way you choose, it involves marking a starting and an ending position for the selection. All the text between the starting and the ending position becomes selected, following the flow of the text like you would follow it when reading it out loud. This is how almost all Windows programs select text, and is appropriate in most situations.

But in some situations, such as when editing text files with information organized in tables, it is more useful to make a rectangular selection, also called a column selection. A rectangular selection is much like a selection of cells in a spreadsheet application, except that in EditPad you select characters instead of cells.

Before you can make rectangular selections, a few conditions must be met. First, you need to use a fixed-width font such as Courier New. You can change the font in Options|Font. Characters are only properly aligned into columns when using a fixed-width font, a requirement to make rectangular selections.

Second, word wrap must be off. You can turn it off using Options|Word Wrap. When long lines are wrapped, editing a long line will cause the text to be rewrapped. This is not a problem when the selection flows along with the text. But when using rectangular selections, this rewrapping would cause the selection to change.

Many text editors do not support rectangular selections. Those that do, are usually IDEs or editors from the DOS world that do not support word wrapping or variable-width fonts. EditPad is a flexible editor and supports both rectangular selections as well as modern conveniences like word wrap and variable-width fonts, but not at the same time for reasons explained above.

There are two ways to make a rectangular selection. First, you can make it the usual way using the keyboard or the mouse, while keeping the Alt button depressed on the keyboard. Second, you can pick Rectangular Selections from the Block menu. This will invert the meaning of the Alt key while making a selection. After choosing Rectangular Selections in the Block menu, a selection made the normal way will be rectangular, and a selection made while pressing Alt will flow along with the text.

If you did not choose a fixed-width font or turn off word wrap, then the selection will always flow along with the text. If you pick Block|Rectangular Selections from the menu, EditPad Pro will warn you that rectangular selections are presently not possible.

Most commands like copy and paste work just the same when using rectangular selections or text-flow selections. Notable exceptions are Block|Indent and Block|Outdent. Some commands, like Block|Fill Columns only work with rectangular selections.

## Block|Begin Selection

Select Begin Selection from the Block menu to tell EditPad Pro to remember the current position of the text cursor. Doing this will not have any visible effect. After using Block|Begin Selection, you can use Block|End Selection to select text starting at the position remembered by Block|Begin Selection. If you have more than one file open in EditPad Pro, Begin Selection will remember the position separately for each file. Selecting Begin Selection when working with one file does not wipe out the position remembered for any other file.

These commands are very useful when you want to make a very large selection, or if you want to perform a search to find the start and the end of the selection. You can do whatever you want between invoking Begin Selection and invoking End Selection, including selecting part of the text in any way. The position stored by Begin Selection is remembered until you use Begin Selection again.

The Begin Selection command works slightly differently when selections are persistent. Then it is possible that part of the file is already selected, without the text cursor being positioned at the end of the selection. In that case, Begin Selection moves the starting position of the selection to the current position of the text cursor, and nothing is remembered. If the new starting position is further away from the ending position than the old one, the selection will be expanded. If the new starting position is below the ending position when the old one was above it, or vice versa, the selection will be pivoted around the ending position. If the new starting position is inside the old selection, the selection will be shrunk.

If there is no selection, Begin Selection remembers the position for use with End Selection, whether selections are persistent or not.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Block|End Selection

After using Block|Begin Selection, you can select End Selection from the Block menu to select the text between the current position of the text cursor and the position stored by Begin Selection. You can use End Selection as many times as you want after using Begin Selection. The selection will continue to start at the same position, until you use Begin Selection again. If you want to make a rectangular selection, select Block|Rectangular Selections from the menu before selecting End Selection.

The End Selection command works slightly differently when selections are persistent. Then it is possible that part of the file is already selected, without the text cursor being positioned at the end of the selection. In that case, End Selection moves the ending position of the selection to the current position of the text cursor, and nothing is remembered. If the new ending position is further away from the starting position, the selection will be expanded. If the new ending position is above the starting position when the old one was below it, or vice versa, the selection will be pivoted around the starting position. If the new ending position is inside the old selection, the selection will be shrunk.

If there is no selection, Begin Selection remembers the position for use with End Selection, whether selections are persistent or not.

If you use the menu item or toolbar button to invoke this command, it will be invoked on EditPad's main editor, where you edit files. If you press the shortcut key on the keyboard, it will be invoked on whichever editor is showing the text cursor (vertical blinking bar), whether that's the main editor, the search box or the replace box.

## Block|Expand Selection

The Expand Selection command in the Block menu is only useful when selections are persistent. Persistent selections allow you to move the text cursor away from the end of the selection. The Expand Selection command will create a new selection that includes the old selection, and all text between the old selection and the position of the text cursor. The position of the text cursor will be the new ending point of the selection.

The most significant difference between Block|End Selection and Block|Expand Selection is that End Selection will pivot the selection if you've moved the text cursor above the spot where you started making the selection while you ended the selection below the starting position, or vice versa. This causes the old selection to be completely deselected, as the starting position stays put. Only the text between the old starting position and the new ending position is selection. Expand Selection will always expand the selection. If the new ending position is above the old starting position while the old ending position was below it, or vice versa, Expand Selection simply makes the old ending position the new starting position.

Whether selections are persistent or not, you can achieve the same effect by pressing Shift on the keyboard while clicking with the mouse at the position where the new selection should end. The advantage of Block|Expand Selection is that you can assign it a keyboard shortcut, and expand the selection with the keyboard.

## Block|Between Matching Brackets

The Between Matching Brackets command is only available when bracket matching is enabled in the syntax coloring section of the file type configuration for the active file's file type. The syntax coloring scheme determines exactly which brackets are matched. See that section in this help file to learn how bracket matching works in EditPad Pro.

When you invoke the Between Matching Brackets command in the Block menu, EditPad Pro selects the text between the nearest pair of matching brackets that surround the text cursor. If some text was already selected, Between Matching Brackets expands the selection to touch the nearest pair of matching brackets surrounding the selection. The brackets are not included in the selection. If the selection already touches the nearest pair of matching brackets, then Between Matching Brackets expands the selection to include the brackets. Repeating the command expands it to touch the next nearest pair of matching brackets. You can repeat the command as long as there is a pair of brackets surrounding the selection.

## Block|Unselect

Use the Unselect command in the Block menu to remove the selection without moving the text cursor. Only the selection is removed. The text that was selected remains.

## Block|Go to Beginning

Select Go to Beginning in the Block menu or press the corresponding keyboard shortcut to quickly move the text cursor to the start of the selection. This command is only useful when selections are persistent, and you've moved the text cursor away from the selection.

## Block|Go to End

Select Go to End in the Block menu or press the corresponding keyboard shortcut to quickly move the text cursor to the end of the selection. This command is only useful when selections are persistent, and you've moved the text cursor away from the selection.

## Block|Insert File

Select Insert File from the Block menu to insert the contents of another file into the file you are currently editing. You will be asked to select the file. The text will be inserted at the current position of the text cursor.If you select more than one file at the same time, their contents will be inserted in alphabetical order of the names of the files

## Block|Write

Select Write from the Block menu to saves the selected portion of the file you are editing into another file. If the other file already exists, EditPad will show a warning and if you confirm, the other file will be overwritten with the selected portion of the current file. If you want to add the selection to the file rather than overwrite it, use Block|Append instead.

If you do not specify an extension for the filename, EditPad will add one depending on the selection you made in the filters (the bottommost drop-down list) in the Block|Write dialog box. If the active filter is "any file" (*.*), no extension will be added. If any other filter is active, the default extension for that file type will be added to the filename. The default extension is the first extension listed in the Extensions setting of the file type's settings.

## Block|Append

Select Append from the Block menu to saves the selected portion of the file you are editing into another file. If the other file already exists, the selection will be added to the end of that file. If it does not yet exist, it will be created.

## Block|Print

Select Print from the Block menu to print the selection portion of the current file, rather than the entire file. A preview of the printout will be shown first. It will allow you to make some changes like which font to print with, which pages should be printed. You can also access the printer setup through the preview window.

There are two key differences between selecting Block|Print in the menu, versus selecting File|Print and turning on the "selection only" in the print preview. The checkbox works at the level of a single line. If a line is partially selected, it will be printed entirely. Block|Print, however, sends the exact selection to the print preview, and ultimately the printer. If you want to print a rectangular block, Block|Print is definitely more useful.

Syntax coloring will also be printed differently. Since File|Print sends the whole file to the print preview, the printed lines will be colored the same whether you're only printing the selected lines, or you're printing the whole file. Block|Print only sends the selection to the print preview, and the syntax coloring will treat the selection as if it is a complete file. If your selection cuts through syntactic elements, they won't be colored properly in the printout when using Block|Print. In that case, you can turn off the "syntax coloring" in the print preview to print without syntax coloring.

# 7. Mark Menu

## Mark|Go to Next Bookmark

Select Go to Next Bookmark in the Mark menu to move the text cursor to the first bookmark after the cursor in the active file. Repeat this command to move through the bookmarks in the order they have in the file, regardless of the numbers on the bookmarks.

## Mark|Go to Previous Bookmark

Select Go to Next Bookmark in the Mark menu to move the text cursor to the nearest bookmark before the cursor in the active file. Repeat this command to move through the bookmarks in the order they have in the file, regardless of the numbers on the bookmarks.

## Mark|Set Any Bookmark

Use the Set Any Bookmark command in the Mark menu to set a bookmark at the current position of the cursor. The number on the bookmark will be the lowest-numbered bookmark that isn't used yet in the active file. When Mark|Project-Wide Bookmarks is active, the number will be the lowest-numbered bookmark that isn't used yet in the active project.

If all 10 numbered bookmarks have been set, the Set Any Bookmark command sets a numberless bookmark. You can place as many numberless bookmarks as you like. Numberless bookmarks cannot be jumped to directly with the Go to Bookmark X commands. You can jump to them with the Go to Next Bookmark and Go to Previous Bookmark commands.

Unlike the Set Bookmark X commands, the Set Any Bookmark command never removes a bookmark from another location. It always sets a new bookmark if the line the cursor is on doesn't have a bookmark yet. If the line the cursor is on does have a bookmark, that bookmark is removed and no new bookmark is set. Using Set Any Bookmark again on a line that already has a numberless bookmark is the only way to remove individual numberless bookmarks.

Though bookmarks are shown in the left margin, they are placed at character positions. Bookmarks stay with the character they were placed at when you edit the file. If you delete the character, the bookmark shifts to the nearest remaining character.

Bookmarks are not saved into a file when you use File|Save. EditPad Pro does automatically remember the bookmarks you set for all files that are listed in the reopen menu, and all files you added to your favorites. Projects also store the bookmarks for all the files in the project. When opening files by opening a project, the files' bookmarks are restored from the project file. When opening a file without opening a project, its bookmarks are retrieved from the reopen menu or the favorites if the file is listed there, even if you didn't use the reopen menu or the favorites menu to open the file.

## Mark|Project-Wide Bookmarks

By default, bookmarks are separate for each file. You can use Mark|Set Bookmark X to set the same bookmark in each file. Mark|Go to Bookmark X will only jump to bookmarks in the current file. The Mark menu will list the bookmarks for the current file.

If you select the Project-Wide Bookmarks item in the Mark menu to make bookmarks project-wide, you will only have one set of 10 numbered bookmarks for all files in the current project. The Mark menu will show the bookmarks for the current project. If you set a bookmark, that bookmark will be set at the position of the text cursor in the current file, and removed from all other files in the project. When jumping to a numbered bookmark, all the files in the current project will be searched through to find the bookmark. The file it was found in will be activated.

If you set the same bookmark in multiple files in the project with project-wide bookmarks turned off, and then turn on project-wide bookmarks, nothing will happen initially. The bookmark will remain set in all the files it is set in. If you try to jump to the bookmark, and it exists in the current file, EditPad Pro will jump to the bookmark in the current file. If not, it will search through the files in the project and jump to the first file it finds. Since this makes it hard to predict where you'll end up if you jump to the bookmark, you should set it again after turning on project-wide bookmarks. Setting the bookmark again will then remove it from all the other files in the project, giving it one place to jump to.

Whether to use separate bookmarks for each file or project-wide bookmarks depends on your personal editing style, and the task at hand. When working with a handful of very large files, using separate bookmarks gives you 10 numbered bookmarks for each file. You can use the tabs to switch between files. When you're working with a very large number of files and you repeatedly need to switch between them, project-wide bookmarks allow you to instantly switch to a particular spot in a particular file.

Regardless of the state of the Project-Wide Bookmarks option, you can use the Mark|Set Any Bookmark command to set as many numberless bookmarks in as many files as you like. The Project-Wide Bookmarks option only affects numbered bookmarks, of which you can have no more than 10.

## Mark|Go to Bookmark 1, 2, 3...

You can quickly jump to any bookmark by pressing Ctrl+1, Ctrl+2, Ctrl+3, etc... on the keyboard. Jumping to a bookmark positions the text cursor on the line on which you set the bookmark with the given number through Mark|Set Bookmark. If the requested bookmark does not exist, nothing happens.

If you remember you set a bookmark but don't remember which number, open the Mark menu. The 10 Go to Bookmark X items in the menu will display the first few words on the line on which each of the bookmarks is set.

Bookmarks can be either separate for each file, or project-wide. When bookmarks are project-wide, the Go to Bookmark command searches for the bookmark in all files in the active project. If the bookmark is found in another file, it will activate that file. When bookmarks are separate for each file, the Go to Bookmark command will never switch to another file.

## Mark|Set Bookmark 1, 2, 3...

You can place up to ten numbered bookmarks in every file open in EditPad Pro using the ten Set Bookmark X commands in the Mark menu. You can quickly do so by pressing Shift+Ctrl+1, Shift+Ctrl+2, etc. on the keyboard. When you place a bookmark, you will see a green square with a white number in the left margin, indicating the bookmark's position. You can remove a bookmark by setting the bookmark again on the same paragraph. Setting a previously set bookmark removes it from its previous location.

If there is already a bookmark on the line the cursor is on, that bookmark is removed. If that bookmark had the same number as the one you're setting, then no bookmark is set. If the removed bookmark was numberless or had a different number, it is replaced with the bookmark number of the Set Bookmark X command you've invoked.

Though bookmarks are shown in the left margin, they are placed at character positions. Bookmarks stay with the character they were placed at when you edit the file. If you delete the character, the bookmark shifts to the nearest remaining character.

Bookmarks are not saved into a file when you use File|Save. EditPad Pro does automatically remember the bookmarks you set for all files that are listed in the reopen menu, and all files you added to your favorites. Projects also store the bookmarks for all the files in the project. When opening files by opening a project, the files' bookmarks are restored from the project file. When opening a file without opening a project, its bookmarks are retrieved from the reopen menu or the favorites if the file is listed there, even if you didn't use the reopen menu or the favorites menu to open the file.

## Mark|Remove All Bookmarks

The Remove All Bookmarks command in the Mark menu removes all the bookmarks from the active file. If Mark|Project-Wide Bookmarks is enabled, it also removes all the bookmarks from all the other files in the active project.

If you want to remove a single bookmark, place the cursor on the line that the bookmark is on and use the Mark|Set Any Bookmark command.

# 8. Fold Menu

## Fold|Fold

Select Fold in the Fold menu to fold the selected lines. The first line in the selection will remain visible, while the others will be hidden. They are "folded" underneath the first line. A square with a plus symbol in it will appear to the left of the folded line. If you click it, the hidden lines will become visible again. A square with a minus will mark the first line, with a vertical line extending down from it to mark the previously folded range. Click the minus button again to refold the range.

If you did not select part of the text, and the text cursor is inside a foldable range indicated by a vertical line in the left margin, the Fold command will fold that folding range.

While folded lines are invisible, they are still fully part of the file, and still take part in all editing actions. E.g. if you select a block that includes one or more folded sections and copy the block to the clipboard with Edit|Copy, all selected lines, including lines hidden by folding, will be copied to the clipboard. Folding only affects the display. The only editing commands that take the folding into account are Edit|Delete Line, Fold|Copy Visible Lines and Fold|Delete Folded Lines.

On the Navigation page in the file type configuration, you can choose if and how EditPad Pro should add automatic folding points. Automatic folding points appear as an unfolded range that you can fold with the mouse or the Fold command.

## Fold|Unfold

Select Unfold in the Fold menu to unfold a section previously folded with Fold|Fold. The Unfold command will only become enabled when you've placed the text cursor on the first (still visible) line of a folded section. The unfolded section will remain marked as a folding point, so you can easily re-fold it.

If you've made a selection, then all folding points inside the selection will be unfolded.

If the current file type uses automatic folding points, then unfolded ranges will automatically disappear when you edit the file. They will be replaced by new automatic folding points. Folded ranges will remain folded until you unfold them.

## Fold|Toggle Fold

If the text cursor is inside a foldable range (as indicated in the left margin), the Toggle Fold command in the Fold menu unfolds the foldable range if it was folded, and folds it it wasn't folded. This is the same as clicking the plus or minus button in the left margin.

If the text cursor is not inside a foldable range, and some text is selected, the Toggle Fold command folds the selected text by creating a new folding range, just like the Fold command in the Fold menu does.

## Fold|Fold Unselected

The Fold Unselected command in the Fold menu folds all existing folding ranges that do not span any selected text. If you did not select any text, the Fold Unselected command folds all folding ranges that do not contain the text cursor.

The Fold Unselected command does nothing with the folding ranges that are (partially) selected. Use Fold|Unfold to unfold those. Using Fold|Fold Unselected followed by Fold|Unfold makes it easy to see the structure of the whole file while focusing on the part of the file that you're editing.

## Fold|Remove Fold

Select Remove Unfold in the Fold menu to unfold a section previously folded with Fold|Fold, and delete the folding point in the process. The Remove Fold command will only become enabled when you've placed the text cursor on the first (still visible) line of a folded section.

If you've made a selection, then all folding points inside the selection will be removed.

If the current file type uses automatic folding points, then automatically added folding points that you removed will automatically reappear when you edit the file.

## Fold|Go to Next Fold

Select Go to Next Fold in the Fold menu to move the text cursor to the first line of the nearest folding range after the cursor. This folding range remains folded or unfolded, as it was.

If the current file type uses automatic folding points, then automatically added folding points that you removed will automatically reappear when you edit the file.

## Fold|Go to Previous Fold

Select Go to Previous Fold in the Fold menu to move the text cursor to the first line of the nearest folding range before the cursor. This folding range remains folded or unfolded, as it was.

## Fold|Fold All

The Fold All command in the Fold menu will fold all automatic folding points, and any unfolded section still marked with a folding point.

## Fold|Unfold All

The Unfold All command in the Fold menu unfolds all sections that you have folded with the Fold command.

## Fold|Toggle All Folds

If some of the foldable ranges in the file are folded, then the Toggle All Folds command in the Fold menu unfolds all those ranges, just like Fold|Unfold All does. If all of the foldable ranges in the file are already unfolded, then the Toggle All Folds command folds them all, just like Fold|Fold All does.

## Fold|Remove All

The Unfold All command in the Fold menu removes all folding points that you added with the Fold command.

## Fold|Copy Visible Lines

The Copy Visible Lines item in the Fold menu copies the selected text to the clipboard. There are two key differences between the Copy Visible Lines command, and the regular Copy command in the Edit menu.

If the selection is not rectangular, and certain lines are only partially selected, the Copy Visible Lines command will copy those lines entirely. The Copy command copies partially selected lines partially.

If certain lines were folded, the Copy Visible Lines command will not copy the lines that were made invisible by the folding. The regular Copy command does copy them.

Text copied by the regular Copy command does preserve folding ranges when pasted back into EditPad Pro. All the text will be pasted, and will remain folded under the pasted folding range. If you paste into another application, then all the text appears without the folding range.

Text copied by the Copy Visible Lines command is always pasted without any folding ranges.

## Fold|Delete Folded Lines

Select the Delete Folded Lines item in the Fold menu to delete all lines in the current selection that were made invisible by folding them.

# 9. Tools Menu

The Tools menu lists all the tools that are defined for the file type of the active file. Select one from the menu and it will be run immediately.

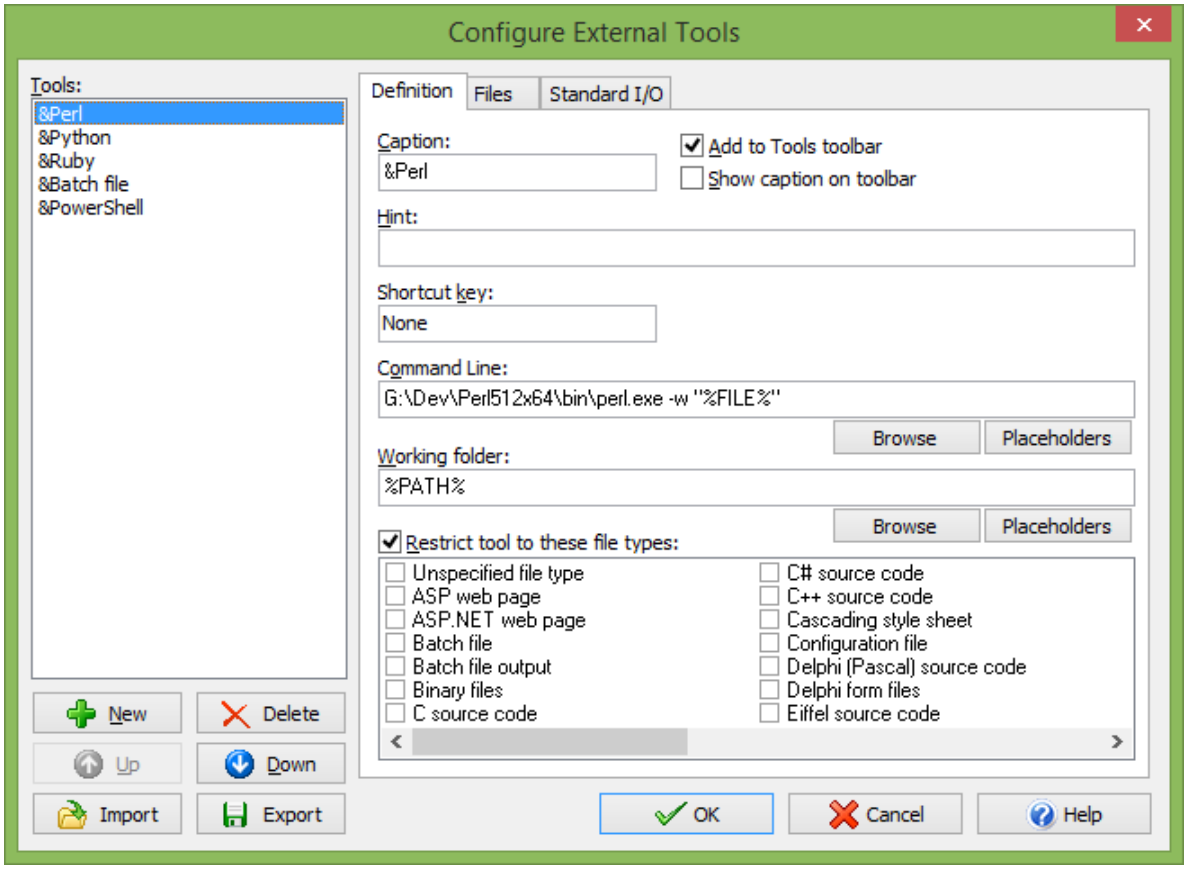
Use the Configure Tools item to configure the tools to be added to the Tools menu. If you have not yet defined any tools, this will be the only choice you can make in the Tools menu.

Tools can be configured to display their output in a message panel inside EditPad Pro. After running a tool you can use the Message Panel item in the Tools menu to show or hide this panel.

## Configure Tools

EditPad Pro has the capability to run any external command or tool. A tool can be any executable program or application, a script run by an interpreter, or even a URL.

To configure tools, select Configure Tools in the Tools menu. The tool configuration screen will appear. The screen is divided into two parts. At the left hand, you'll see a list of currently defined tools. If you haven't defined any tools yet, this list will be empty. At the right hand are three tabs that hold the settings for the tool that's currently selected in the list at the left. The tabs don't appear until you've added at least one tool.



To change a tool's settings, simply click on it in the list and make the changes you want. To create a new tool, click the New button below the list and set the options as you want them. To delete a tool, click on it and click the Delete button. The order of the tools in the list is the order in which they will be shown in the Tools menu. Select a tool and click the Up or Down button to move it.

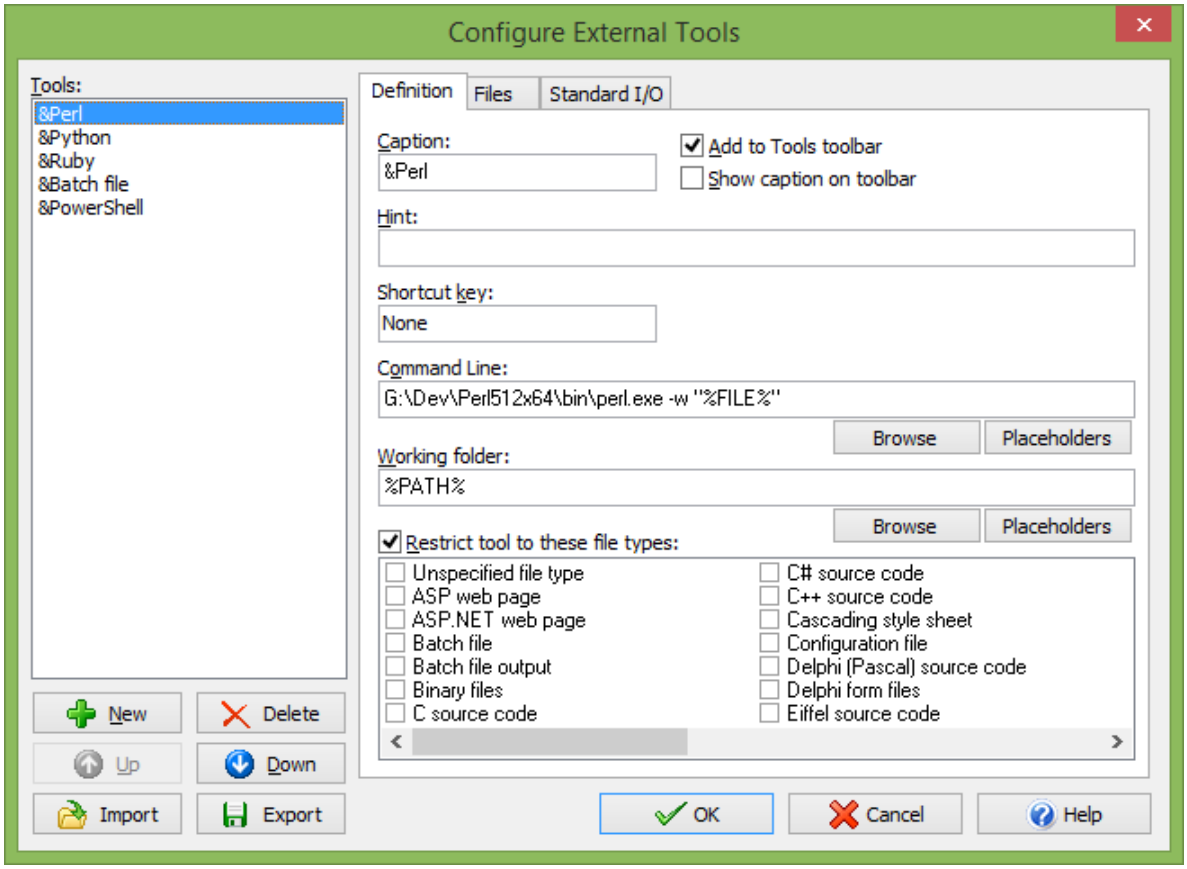
You can select multiple tools by holding down the Shift or Control key on the keyboard while clicking in the list of tools. The controls under the tabs will indicate the settings for all tools. If an edit box or drop-down list shows a value, that means all the selected tools use the same value. Otherwise, the edit box will be blank. If a checkbox is checked or cleared, that means it is checked or cleared for all the selected tools. Otherwise, the checkbox will be filled with a square. If you make a change to any setting, that change is applied to all the selected tools.

You can save tool configurations into a file that you can share with other people. To save one or more tools, select them in the list and click the Export button below the list. You'll be asked for a name of the .ini file into which all the tools will be saved. To load a tool configuration file you've received from somebody, click the Import button. The loaded tools will be added to the list.

For each tool, you can specify three sets of options:
Definition: Basic settings you need to make for every tool, so EditPad Pro knows which tool to run.
Files: Options for opening and saving files to be passed on the tool's command line.
Standard I/O: Options for transferring text to and from console applications. Console applications are applications that use a textual interface or DOS box rather than a graphical interface like EditPad Pro.

## Tool Definition

When adding a tool to EditPad Pro's Tools menu, there are three pages of settings that you can make for each tool. On the "Definition" page you can set the basic options needed to add any application to EditPad Pro's Tools menu.




"Caption" is the caption that this tool's menu item in the Tools menu will have. If you want to make a certain character a hotkey in the menu, precede it with an ampersand (&). The character that follows the ampersand will be shown underlined in the menu (unless you disabled the underlining of hotkeys in Windows).

Check "add to Tools toolbar" if you want a button for the tool to appear on the Tools toolbar. This toolbar is hidden by default. You can make it visible by right-clicking on the main menu or any toolbar and selecting Tools.

Check "show caption tool toolbar" if you want the toolbar button for the tool to show the caption that you specified. If not, only the tool's icon is shown. The icon is automatically loaded from the file or application you specify on the command line. Showing the caption can be useful if you have multiple tool configurations for the same application. It does make the tool take up much more space on the toolbar. Items in the Tools menu always show the caption.

"Hint" is the text that will be shown in the status bar when the mouse is pointing at this tool's menu item in the Tools menu. You may leave this blank if the caption is descriptive enough.

If you will use the tool often, you can assign a keyboard shortcut key combination to it. Click on the "shortcut key" field, and then press the shortcut key combination you want on the keyboard. E.g. if you press F2 on the keyboard, the "shortcut key" field will display F2. The menu item for the tool in the Tools menu will also indicate F2. When you press F2 on the keyboard while working with EditPad Pro, the tool will be run. To remove the shortcut from the tool, click in the "shortcut key" field and press the backspace key on the keyboard.

For the "Command Line", you need to type in what you would type in if you would launch the tool from the Run item in the Windows Start menu. It is best to include the full path to the executable and any file parameters. Note that you must *not* use <, > or | characters for file redirection. If you need file redirection, use the Standard I/O page. If the path to the tool's executable has spaces in it, you must enclose it in double quotes. Click the Browse button to browse for the executable file.

If you specify a document file instead of an executable file as the command line, then EditPad Pro will launch the document's associated application, and pass the document file as a parameter. Windows Explorer does the same when you double-click on a document.

If you specify a URL as the command line, then EditPad Pro opens the URL in the web browser configured in the System Preferences.

"Working Folder" is the folder that will appear to be the current one to the tool when it is run by EditPad Pro.

You can use special path placeholders to use the full path to the file you are currently editing, or parts of the file name or path in the command line or in the working folder. If you have marked any of the four temporary file options (see below), you can also use placeholders for the path and file name of the temporary file. There are also path placeholders that allow you to select one of the files open in EditPad Pro or a file on disk to be passed on the command line. Note that many applications cannot handle file names with spaces in them on the command line, unless they are enclosed by double quotes. So it is best to always put double quotes around paths you compose using path placeholders, like I did in the above screen shot.

Click the Placeholders button to get a screen listing all path placeholders, making it easy to use them on the command line. The screen will use the file and project you currently have open in EditPad as example paths. If the active file or project are untitled, the example will be a dummy path. The temporary file placeholders will only be available if you've turned on at least one of the options to open or save a temporary file.

When running EditPad Pro from a removable drive, the drive will get a different drive letter each time it is inserted into another computer. In that situation, you should reference other applications stored on the same device using the %EPPDRIVE% and %EPPPATH% path placeholders. Then your tools will always reference the files on your drive, regardless of the drive letter it gets.

Since most tools will only make sense when used on a file of a certain type, you can select the file types this tool applies to in the list at the right. Do this by marking the appropriate checkboxes. Not marking any checkbox has the same effect as marking all of them. The Tools menu will show all the tools that apply to the type of file you are currently editing.

## Command Line Placeholders for Tools

EditPad Pro supports a range of placeholders that you can use on a tool's command line to pass information about or parts of the file you're editing in EditPad Pro.

%POS%: Number of bytes before the position of the text cursor. Depending on the file's encoding, the number of bytes may not equal the number of characters.
%LINE%: Number of the line the text cursor is on. %COL%: Number of the column the text cursor is on. %LINETEXT: Text of the line the text cursor is on. %CHAR%: The character immediately to the right of the text cursor. %WORD%: The word under the text cursor. %SELSTART%: Number of bytes before

the start of the selection. %SELSTOP%: Number of bytes before the end of the selection. Subtracting %SELSTART% from %SELSTOP% gives the length of the selection in bytes. %ENCODING%: Encoding used by the active file, such as `utf-8`, `windows-1252`, `iso-8859-1`, etc.

## Path Placeholders

You can use path placeholders to use part or all of the path and the file name of the file you are currently editing. You can use them in the command line and working folder for tools. This way, you can have a tool work with the file you are editing in EditPad Pro. In the examples below, the file being edited is C:\data\files\web\log\foo.bar.txt. If the current file is untitled, all the placeholders below will be replaced with nothingness.

If you want to use the path or name of the temporary file that EditPad Pro can open or save when running a tool, simply prepend TEMP to the name of the placeholder. E.g. %TEMPFILE% is the full path and filename to the temporary file. %TEMPPATH% is the folder the temporary file is in.

If the file was opened via FTP, use %FTPFILE% to specify the full path to the file on the FTP server. %FTPSERVER% is the domain name of the server, and %FTPURL%: the full `ftp://` URL to the file.

If you want to pass more than one file on the tool's command line, you can use %PICK1FILE% through %PICK9FILE% to specify (parts of) the paths to up to 9 files that you have already open in EditPad Pro. If you use these placeholders, each time you run the tool a window will pop up for you to select the file(s) to be passed on the command line from the files you have open in EditPad Pro. The files you select will be saved automatically so the tool can get their current contents. If you don't want the files to be saved, use %TEMP1FILE% through %TEMP9FILE% instead. These placeholders show the same file picker for files open in EditPad Pro. They don't save the files but save temporary copies instead. The paths to the temporary copies are passed to the tool.

If you want to pass any file, regardless of whether it is open in EditPad Pro or not, use %DISK1FILE% through %DISK9FILE%. These placeholders show a regular open file dialog so you can select a file from disk. If you select a file that is open in EditPad Pro, any unsaved changes are not saved.

All the %FILE% placeholders described in this section represent a whole series of placeholders that hold various parts of the file's path.

| **Placeholder** | **Meaning** | **Example** |
|---|---|---|
| %FILE% | The entire path plus filename to the file | C:\data\files\web\log\foo.bar.txt |
| %FILENAME% | The file name without path | foo.bar.txt |
| %FILENAMENOEXT% | The file name without the extension | foo.bar |
| %FILENAMENODOT% | The file name cut off at the first dot | foo |
| %FILEEXT% | The extension of the file name without the dot | txt |

| | | |
|---|---|---|
| %FILELONGEXT% | Everything in the file name after the first dot | bar.txt |
| %PATH% | The full path without trailing delimiter to the file | C:\data\files\web\log |
| %DRIVE% | The drive the file is on, without trailing delimiter | C: for DOS paths; \\server for UNC paths; blank for UNIX paths |
| %FOLDER% | The full path without the drive and without leading or trailing delimiters | data\files\web\log |
| %FOLDER1% | First folder in the path | data |
| %FOLDER2% | Second folder in the path | files |
| (...etc...) | | |
| %FOLDER99% | 99th folder in the path. | *(nothing)* |
| %FOLDER<1% | Last folder in the path | log |
| %FOLDER<2% | Second folder from the end in the path | web |
| (...etc...) | | |
| %FOLDER<99% | 99th folder from the end in the path. | *(nothing)* |
| %PATH1% | First folder in the path, without delimiters | data |
| %PATH2% | First two folders in the path, without leading or trailing delimiters | data\files |
| (...etc...) | | |
| %PATH99% | First 99 folders in the path, without leading or trailing delimiters | data\files\web\log |
| %PATH<1% | Last folder in the path, without delimiters | log |
| %PATH<2% | Last two folders in the path, without leading or trailing delimiters | web\log |
| (...etc...) | | |
| %PATH<99% | Last 99 folders in the path, without leading or trailing delimiters | data\files\web\log |
| %PATH-1% | Path without the drive or the first folder | files\web\log |
| %PATH-2% | Path without the drive or the first two folders | web\log |
| (...etc...) | | |
| %PATH-99% | Path without the drive or the first | *(nothing)* |

| | | |
|---|---|---|
| | 99 folders. | |
| %PATH<-1% | Path without the drive or the last folder | data\files\web |
| %PATH<-2% | Path without the drive or the last two folders | data\files |
| (...etc...) | | |
| %PATH<-99% | Path without the drive or the last 99 folders. | *(nothing)* |

## Combining Path Placeholders

You can string several path placeholders together to form a complete path. If you have a file c:\data\test\file.txt then d:\%FOLDER2%\%FILENAME% will be substituted with d:\test\file.txt. However, if the original file is c:\more\file.txt then the same path will be replaced with d:\\file.txt because %FOLDER2% is empty. The result is an invalid path.

The solution is to use combined path placeholders, like this: d:\%FOLDER2\FILENAME%. The first example will be substituted with d:\test\file.txt just the same, and the second will be substituted with d:\file.txt, a valid path. You can combine any number of path placeholders into a single path placeholder, separating them either with backslashes (\) or forward slashes (/). Place the entire combined placeholder between two percentage signs.

A slash between two placeholders inside the combined placeholder is only added if there is actually something to separate inside the placeholder. Slashes between two placeholders will never cause a slash to be put at the start or the end of the entire resulting path. In the above example, the backslash inside the placeholder is only included in the final path if %FOLDER2% is not empty.

A slash right after the first percentage sign makes sure that the resulting path starts with a slash. If the entire resulting path is empty, or if it already starts with a slash, then the slash is not added.

A slash right before the final percentage sign makes sure that the resulting path ends with a slash. If the entire resulting path is empty, or if it already ends with a slash, then the slash is not added.

Mixing backslashes and forward slashes is not permitted. Using a forward slash inside a combined placeholder, will convert all backslashes in the resulting path to forward slashes. This is useful when creating URLs based on file names, as URLs use forward slashes, but Windows file names use backslashes.

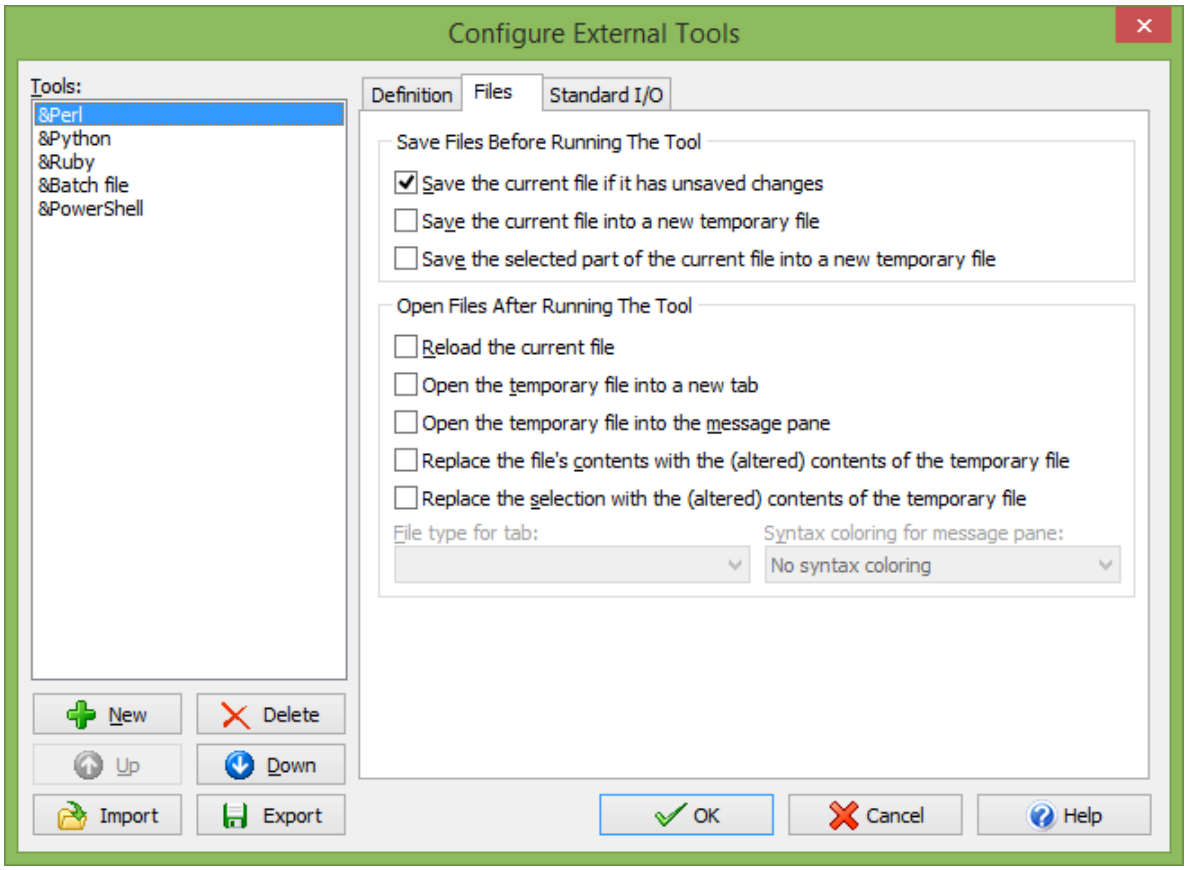
Example: If the original path is c:\data\files\web\log\foo.bar.txt

%\FOLDER1\% => \data\
%\FOLDER5\% => (nothing)
%PATH-2\FILENAME% => web\log\foo.bar.txt
%PATH-2/FILENAME% => web/log/foo.bar.txt
%PATH-4\FILENAME% => foo.bar.txt
%DRIVE\PATH-2\FILENAME% => c:\web\log\foo.bar.txt
%DRIVE\PATH-4\FILENAME% => c:\foo.bar.txt

%\FOLDER1\FOLDER4\% => \data\log\
%\FOLDER1\FOLDER5\% => \data\

## Tool Files

When adding a tool to EditPad Pro's Tools menu, there are three pages of settings that you can make for each tool. On the "Files" page you can choose if EditPad Pro should save the current file or selection, or open files modified by the tool.




Mark "save the current file if it has unsaved changes" if EditPad Pro should automatically do a File|Save before the tool is run. If the file is untitled, you will be prompted for a file name. You can use the %FILE% path placeholder to pass the file on the tool's command line.

Mark "save the current file into a new temporary file" if EditPad Pro should automatically do a File|Save Copy As and save the file under a new, temporary name before the tool is run. This is useful if the tool will modify the file's contents but you do not want to lose the original, or if you want to test changes you made to a file with a tool without having to save those changes first. The temporary file will be deleted after the tool has finished running.

Mark "save the selected part of current file into a new temporary file" if EditPad Pro should automatically do a Block|Write before the tool is run. You can use this to allow the tool to read or modify the selection. You cannot use this option in combination with saving the entire file into a temporary file, but you can use it in

combination with both “open temporary file” (see above) and “replace selection” (see below). The temporary file will be deleted after the tool has finished running.

Turn on “reload the current file” to force EditPad Pro to reload the file that was active when you invoked the tool as soon as the tool has finished running. You should do so if the tool will modify the current file, even if you’ve turned on the option to reload modified files in the Open Files Preferences. Otherwise, EditPad Pro may not notice the file has changed until you switch between tabs in EditPad or between EditPad and another application.

Mark “open the temporary file into a new tab” if the tool will create or modify the temporary file and you want to see the changes in EditPad. If this option is checked, EditPad will open the file into a new tab and then delete the temporary file from disk. If you want to save the tool’s result, use File|Save As. You can select the file type that EditPad Pro should use for the file. Select “same as current file” if the tool doesn’t output a specific file type.

Mark “open the temporary file into the message panel” if the tool will create or modify the temporary file and you want to see the changes in a side panel in EditPad. If this option is checked, EditPad will open the file into the message panel and then delete the temporary file from disk. You can select the syntax coloring scheme that the message panel should use. This scheme can be a scheme that is not used by any file type.

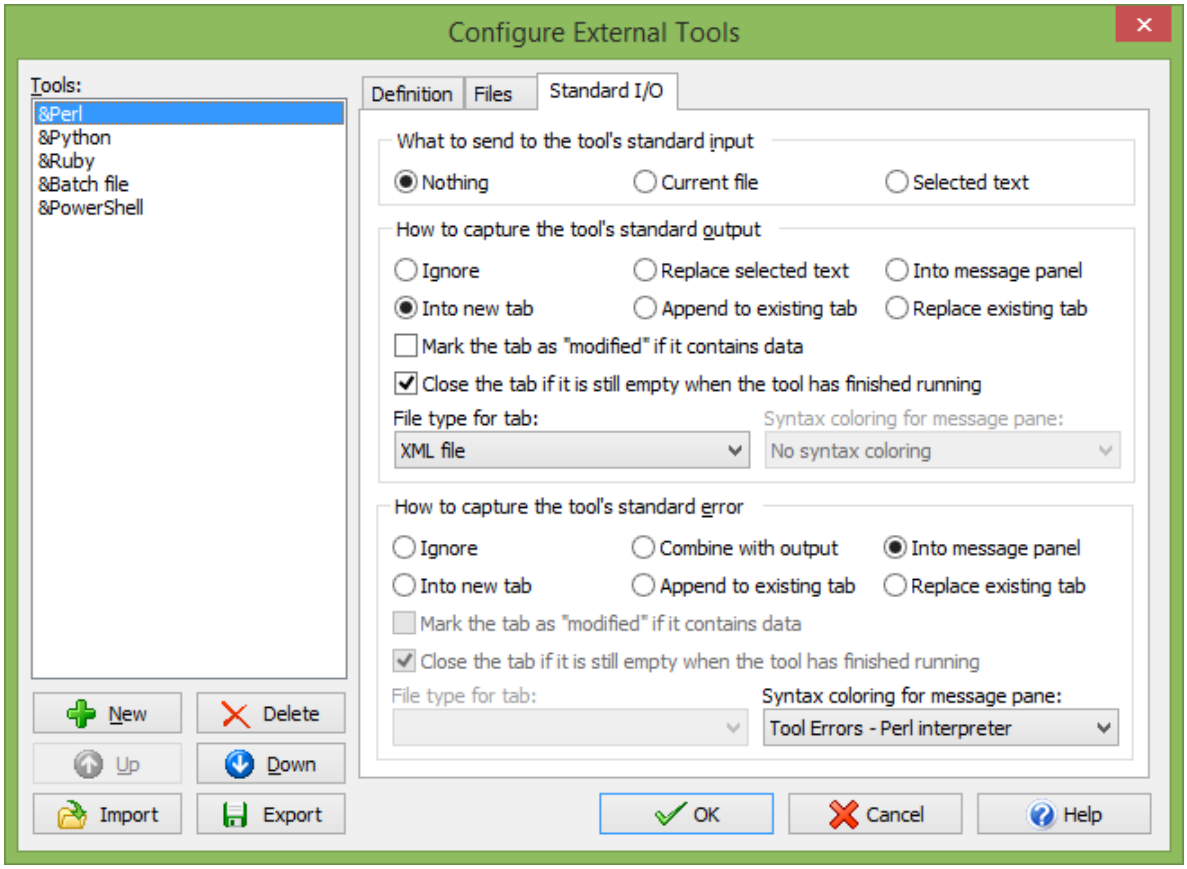
Mark "replace the file’s contents with the (altered) contents of the temporary file" to make EditPad Pro delete all text in the active file and then do a Block|Insert File using the temporary file, after the tool has finished running.

Mark "replace the selection with the (altered) contents of the temporary file" if EditPad Pro should automatically delete the current selection and do a Block|Insert File using the temporary file, after the tool has finished running. Together with the “save the selected part of current file into a new temporary file” option, you can add a tool to EditPad Pro’s menu that modifies the current selection. You could write a script to add special functionality that EditPad Pro does not provide.

If you’ve turned on at least one of the options to either open or save a temporary file, you can use the %TEMPFILE% path placeholder to pass the temporary file’s name on the tool’s command line. All of the options will work on a single temporary file for each tool.

## Tool Standard I/O

When adding a tool to EditPad Pro’s Tools menu, there are three pages of settings that you can make for each tool. If you are adding a console or text mode application to EditPad Pro’s Tools menu, then you can instruct EditPad Pro to communicate with the application via the options on the "Standard I/O" page.




By default, console applications read input from the keyboard and write output to the screen. On the DOS and Windows command line, you can use the less than and greater than symbols to redirect standard input and output. The command line "sometool <input.txt >output.txt" will cause sometool to read from input.txt and write to output.txt.

You cannot use these symbols on the command line when configuring EditPad Pro to run a console application. However, you do have a variety of options for redirecting standard I/O on the "Standard I/O" page.

### What to Send to Standard Input

If a tool expects data via standard input (i.e. it expects a DOS/Windows command line like "sometool <input.txt"), EditPad Pro is capable of sending either the current file entirely, or only the selected part of the current file. The current file is whichever file you're editing at the moment you run the tool by selecting it in the Tools menu.

If the tool is a text-based console application that accepts input from the keyboard, anything EditPad sends to the tool's standard input will be processed by the tool as if you typed it on the keyboard. This does not work with graphical applications, since those do not use standard input to retrieve keyboard input.

Before running the tool, EditPad Pro makes an internal copy of the text it needs to send to standard input. If the tool takes a long time to run, editing or closing the file EditPad is sending to the tool will not affect the tool's execution.

## How to Capture Standard Output

If a text-based console application writes its output to the screen, or if an application outputs data to a file with a command line like "sometool >output.txt", EditPad Pro can capture that output in various ways:

- Ignore: The tool's standard output will be lost. Select this option if the tool doesn't output anything useful to the screen, or if it's a graphical application that doesn't support standard output.
- Replace selected text: EditPad will replace the selected part of the file that was active when you invoked the tool with the tool's output. If there is no selection, the tool's output will be inserted at the text cursor position. This option is particularly useful in combination with the option to send the selected text to standard input. That way, an external tool can provide text processing functions not provided by EditPad Pro itself.
- Into message pane: EditPad will capture the tool's output into a separate pane. Since EditPad Pro has only one message pane, the output of any previous tools that was captured into the message pane will be lost. Choose this option when a tool outputs status or error messages that you want to inspect but not keep. The message pane supports syntax coloring. EditPad Pro ships with a few "Tool Errors" and "Tool Output" syntax coloring schemes that make the output of some popular tools a bit more readable. You can use the Syntax Coloring Scheme Editor to create your own schemes, or you can download schemes in the Colors and Syntax tab in the File Type Configuration.
- Into new tab: EditPad will create a new file tab to capture the tool's output. If you run the tool more than once, a new tab will be created each time. Select this option if the tool outputs a lot of data or a complete file. You can edit, save and close the new tab like any other file you'd edit in EditPad.
- Append to existing tab: EditPad will create a new tab to capture the tool's output, and continue using that tab each time you run that tool, appending additional data each time. Choose this option if you want to save the results of multiple invocations of a tool into a single file.
- Replace existing tab: EditPad will create a new tab to capture the tool's output, and replace that tab's contents whenever you run the tool again. Make this choice if the tool outputs too much information to neatly fit into the message pane, and you only want to keep the results of the last invocation of the tool.

If you choose one of the three options to capture the output into a tab, you can make three additional settings for that tab:

- Mark the tab as "modified" if it contains data: Turn on this option if you usually want to save the captured output into a file. If you close the file without saving, EditPad Pro will prompt just like when you close an unsaved file. Turn off this option if you usually want to discard the output. Note that if you edit the output, the tab will still become marked as modified.
- Close the tab if it remains empty: This option is particularly handy when you've chosen the "into new tab" capturing option. It automatically closes the tab if nothing was captured, so your workspace doesn't get cluttered with empty tabs.
- File type for tab: Since standard output doesn't have a file name, EditPad Pro cannot use the file masks from the file type configuration to determine the output's file type. Therefore, you can select one of the file types from the list. Choose the "same as current file" option for text processing tools that return a modified version of the file you're editing.

If you close the tab EditPad Pro is using to capture standard output while the tool is still running, the tool's output will be lost. However, the tool will keep running until normal completion. EditPad Pro will keep clearing the tool's standard output without saving it anywhere.

### How to Capture Standard Error

Standard error works just like standard output. While standard output is intended for console applications to produce their output, standard error is intended for their error messages. Not all console applications respect this distinction. Some may send error messages to standard output. When the application is run from a command prompt with output appearing on the screen, standard output and error are automatically combined onto the single screen. This makes some programmers forget to make their console applications distinguish between standard output and error.

Therefore, EditPad Pro offers the "combine with output" option. EditPad Pro will then mingle standard error with standard output, just like a console screen would. The combined output will end up in whichever location (tab or message pane) you selected for standard output.

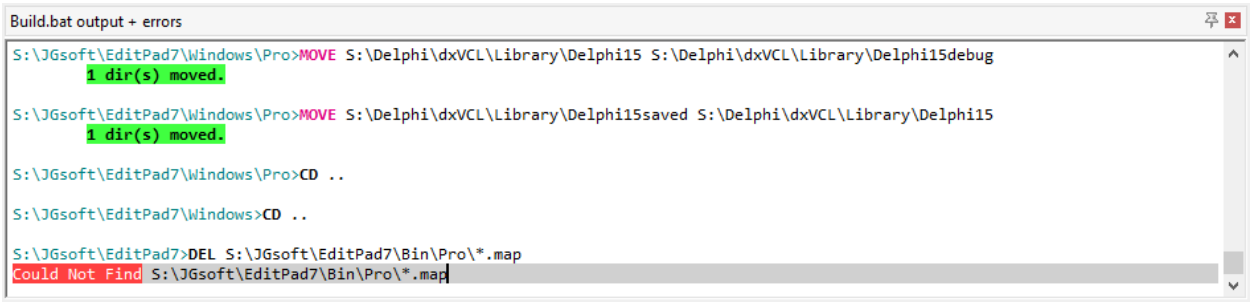
All the other choices for standard error are identical to those for standard output. If you select one of the three tab options for both standard output and standard error, you will get two new tabs: one for standard output, and one for standard error.

Since EditPad Pro has only one message pane, the only way to capture both standard output and standard error into the message pane is by selecting the "message pane" option for standard output, and the "combine with output" option for standard error.

## Tools | Message Panel

On the Files tab in the tool configuration you can choose to load a temporary file created or modified by a tool into the Message panel. On the I/O tab you can choose to have the tool's standard output and/or error redirected to the Message panel. The Message panel is a side panel in EditPad Pro. It has just one read-only edit box for displaying tool output. You can right-click the edit box to select and copy the text, and to toggle word wrap.

If you've closed the Message panel after running a tool, you can use the Message Panel item in the Tools menu to make it visible again. This item only appears in the Tools menu after you've run a tool that uses the Message Panel. You cannot make the Message Panel visible if it hasn't been used by a tool yet.



# 10. Macros Menu

Via the Macros menu you can record and play back keystroke macros. With keystroke macros, you can automate repetitive editing tasks.

Macros that you have already recorded appear at the top of the Macros menu. Selecting a macro in the menu executes it once. Select Organize Macros in the Macros menu if you want to run a macro multiple times.

## Macros|Record Macro

To record a macro, select the Record Macro item in the Macros menu. EditPad Pro will ask you to enter a name for the macro. When you confirm the macro's properties, recording begins immediately.

When you're done recording the macro, select the Stop Recording item in the Macros menu. You'll notice it's the only available command during recording. You cannot play back and record macros at the same time.

To play back a macro, simply select it from the Macros menu or press its keyboard shortcut. The macro will be executed once. Select Organize Macros in the Macros menu if you want to run a macro multiple times.

EditPad Pro automatically saves macros. You won't be prompted for a file name. If you want to save a macro into a file so you can share it with other people, use the Export command in the Organize Macros screen.

## How EditPad Pro Records Macros

Before you start recording and playing back macros, it's important that you understand exactly what EditPad Pro records. If you understand macros, they're a powerful and quick way to automate otherwise tedious and repetitive editing tasks. If you don't understand macros, they're a quick way to completely mess up your files.

### Mouse Clicks Are Not Recorded

Clicks are relative to the position and size of EditPad's windows, to the location where you've scrolled, etc. A click at a certain position on the screen may have a totally different meaning from one situation to the next. Therefore, mouse clicks are not recorded. If you move the mouse over the text editing area, the mouse pointer will indicate you can't click. If you try, the click will have no effect at all.

Menu item commands and button commands are recorded. You can invoke them with the mouse while recording a macro. EditPad Pro will record the command that you clicked on, rather than the click itself.

### Only Your Actual Actions Are Recorded

EditPad Pro only records exactly those keystrokes that you type, or exactly those commands that you invoke when running a macro. E.g. if you select the Search|Find Next menu item while recording a macro, the macro only records that you selected Search|Find Next. It does *not* record the search term, the search options, etc. If you play back the macro, it will simply execute Search|Find Next again. If you've changed the

search term or search options since recording the macro, the new term and options will be used for the search.

By recording only your actual actions, you can make generic macros. E.g. if you want a macro that deletes every line with a search match, you can record Search|Find Next and Edit|Delete Line. You can then use that macro to delete the matches of any search term. Simply enter the correct search term before running the macro.

This also applies to any keystrokes you type into the editor, search box or replace box. The macro will record the actual keystrokes. Their effect may be different when you run the macro, depending on the position of the text cursor and if any text was selected. Keep this in mind when recording macros. You may want to press a few extra keys to make sure the macro works well. E.g. instead of requiring a macro to have the cursor at the start of the line, simply press the Home key when you start recording the macro. Even if the cursor was already at the start of the line when you recorded the macro, it will still record that you pressed the Home key.

### How to Record Search Terms

If you want to record search terms as part of a macro, you can do so by selecting the search term from the Search|History or Search|Favorites menu. The macro will then record the search text, replace text and all search options. You can temporarily add a search term to the favorites before recording the macro. The macro will store the actual search terms, not the fact that you selected an item from the history or favorites. If you later delete the item from the history or favorites, the macro will still work.

### Macros Record Command Options

Some commands in the Extra menu, like Delete Duplicate Lines and Compare Files show a screen with options before doing their work. If you use these commands while recording, the macro will record the state of the options screen when you click the OK button. It will not record whether you toggled any options, but the final result. When you play back the macro, the command will be executed with those options every time, without showing the options screen.

This rule may seem to contradict the "only your actual actions are recorded" rule. However, there's a key difference between the search options and the alphabetic sort options. You can set the search options in EditPad Pro's main window. If you toggle a search option, the option's state is recorded. Therefore, you can "prepare" the search options before playing back a macro that didn't record them. You could even record a macro that does nothing but set search options.

The Delete Duplicate Lines command's options can only be set when you've already told EditPad Pro to delete duplicate lines. You can't prepare the options prior to executing the macro.

The state of options that you can prepare are not recorded by macros, since actions that change the state of options that you can prepare are recorded. Commands with options that you cannot prepare record the state of the options as part of the command.

### Failed and Inconsequential Commands Are Recorded

Pressing the Home key on the keyboard while the text cursor is already at the start of the line does nothing. However, macros still record that inconsequential key press. When the macro is played back, it will move the cursor to the start of the line if it isn't there already.

Search commands are special in EditPad Pro. They can fail. If you click the Find Next button and it flashes, that means it failed. There was no next search match. While recording a macro, the Find Next command is still recorded.

### Failed Commands Halt Macro Playback

Though not exactly a recording issue, it's important to remember that if a search command fails during macro playback, it will stop the macro immediately. This allows you to run macros that run until failure to process all search terms.

## Organize Macros

Select Organize Macros in the Macros menu to organize, play back and record macros.

### Repeated Playback

Playing back a macro via the Organize Macros allows you to play back the same macro many times. Simply enter the number of times the macro should run in the Repeat box, and click the Play Macro button. The macro may run fewer times than you specified, but never more.

If the macro includes a search command, and the search fails to find a match, macro playback will stop immediately. This enables you to record a macro that does something to a search match, and then run that macro on all search matches. Suppose you record a macro that executes the Search|Find Next command and then the Edit|Delete Line command. If you then execute this macro with Repeat set to 100, the macro will delete the lines containing the next 100 search matches. If there are fewer than 100 search matches, all their lines will be deleted, and the macro will stop. You can use the Search|Count Matches command to determine the number of times the macro should run, or you could simply enter a large enough number.

However, there is a reason why you still need to enter an upper limit for the number of times the macro should run. You can't tell EditPad Pro to run the macro until the search fails. The reason is that it's quite possible for a search command to never fail. E.g. a macro that runs Search|Find First but doesn't delete the search match will always find that search match. A macro using Search|Find Next with the "loop automatically" option turned on will loop forever if it doesn't delete search matches. By specifying a reasonable number of times for the macro to repeat itself, you can prevent it from going on forever.

### Recording

Click the Record button to record a new macro. EditPad Pro will ask you to enter a name for the macro. When you confirm the macro's properties, the Organize Macros screen closes and recording begins

immediately. There are a number of important issues to keep in mind when recording macros. Make sure to review them.

When you're done recording the macro, select the Stop Recording item in the Macros menu. You'll notice it's the only available command during recording. You cannot play back and record macros at the same time.

### Organizing Macros

Click the Properties button to rename a macro or to change its shortcut key, or to rename a folder. You can add folders with the New Folder button. To put macros into a folder, simply drag and drop them with the mouse. If you have a lot of macros, organizing them in folders makes it easier to find them later.

### Import and Export

EditPad Pro automatically saves all your macros internally along with your preferences, history lists, etc. If you want to share a macro with other people, or copy it from one computer to another, use the Export Macro button. The selected macro is then saved into the file that you specified. You can open the file in EditPad Pro to view its contents if you like. However, macro files are not intended to be edited.

If you've received a macro that you'd like to run in a file, first import it with the Import Macro button. The macro is then added to your list of macros. You can play back imported macros just like macros that you recorded yourself. The macro file is no longer needed once you've imported it.

You can export multiple macros into a single file by selecting multiple macros before clicking the Export Macro button. If you include one or more folders in your selection, all the macros in those folders and their subfolders will be exported. If you click the Export Macro button a second time and select an existing file, EditPad Pro will replace that file rather than append to it. When importing a file that holds more than one macro, all of the macros will be imported. If you wanted to import only some, you can delete the extraneous ones after importing the file.

## Macros|Record Instant Macro

The Record Instant Macro command in the Macros menu records a macro just like Macros|Record Macro does. The only difference is that EditPad Pro doesn't ask you for a name and keyboard shortcut for the macro. The only way to play back the instant macro is with Macros|Play Instant Macro.

Only one instant macro is preserved at a time. Recording a new instant macro deletes the previous instant macro without warning.

Use instant macros when you want to quickly record and play back a group of keystrokes a few times.

## Macros|Play Instant Macro

Use the Play Instant Macro command in the Macros menu to play back the macro you last recorded with Macros|Record Instant Macro.

## Macro Properties

The Macro Properties screen appears when you select the Record Macro

item in the Macros menu, and when you click the Properties button when organizing macros.

EditPad Pro macros have only two properties: a name, and an optional keyboard shortcut. Each macro must have a unique name. If you start recording a new macro with the same name as an existing macro, the existing macro is automatically and silently replaced.

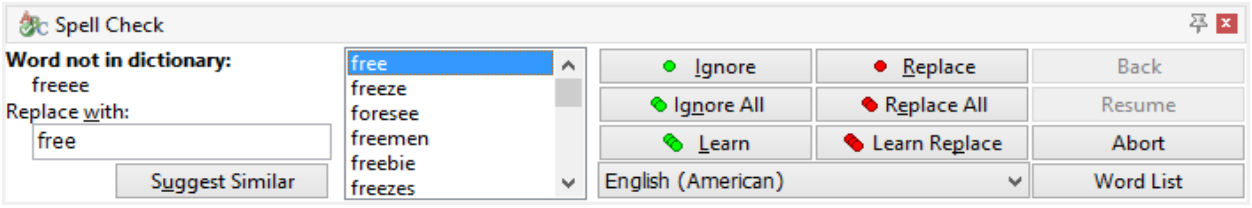
The keyboard shortcut can either be a single key combination, or a double key combination. Assigning a shortcut to a macro works just like assigning shortcuts in the Keyboard preferences. The procedure is explained in detail in the help topic about Keyboard preferences. If you assign a shortcut to a macro that is already assigned to another macro, the new macro will take over the shortcut from the old macro. You cannot assign a shortcut that's already used for a menu item to a macro.

# 11. Extra Menu

## Extra|Spell Check

Select |Spell Check from the Extra menu to check the spelling of the current file. The spell check function always starts from the beginning of the file.

The pane below appears during the spell check process:




When a word is not found in the dictionary, it will be selected in the editor and reported in the upper left area of the spell check pane. EditPad Pro will start searching through the dictionary for words similar to the misspelled one, and show them in the list. As long as "(searching...)" is visible, EditPad Pro is still looking for more possible, correctly spelled substitutions. You do not have to wait until EditPad Pro is finished. You can correct the error right away.

If the word is really misspelled, you can type in the correctly spelled word in the “Replace with” field. You can also click on an item in the list with EditPad Pro’s suggestions and it will automatically be placed in the “Replace with” field. If you click on “Suggest similar”, EditPad Pro will start looking for words in the dictionary similar to what you typed into the “Replace with” field.

Then you need to click one of the three replace buttons: “Replace” will replace the misspelled word with the replacement just this time. “Replace All” will replace the misspelled word with the replacement every time it is found during the current EditPad Pro session. When you pick File|Exit or switch to a different language, the replacement is forgotten. If you click “Learn Replace”, EditPad Pro will always automatically replace the misspelled word with the replacement and remember this even if you quit EditPad Pro.

You can also fix the error directly in the editor. After doing so, you will have to click the “Resume” button to continue to check the spelling of the rest of the file.

If the word is correctly spelled, you can click “Ignore” to accept the word as correct just this time. “Ignore All” will make EditPad Pro ignore all occurrences of this word until you switch languages or close EditPad Pro. “Learn” will make EditPad Pro add the word to its dictionary and will from then on always be accepted as correct.

If you make a mistake, click on “Back” to restore the original word.

Click “Abort” to stop spell checking and close the pane.

The “Word List” button allows you to edit the list of words that you added to the custom dictionary. When you click the button, a new window will appear. The left hand side of the window contains the list of words that you specified as correctly spelt by clicking on the “Learn” button. The right hand side of the window

contains pairs of words separated by an equals sign (=). These are the automatic replacements you added by clicking the "Learn Replace" button. Whenever the spell checker finds the word at the left side of the equals sign, it will be replaced with the word at the right of the equals sign.

## Extra|Spell Check Selection

Select "Spell Check Selection" from the Extra menu to check the selected portion of the current file for spelling errors. See Extra|Spell Check for a description of the spell check pane.

## Extra|Spell Check Project

Select "Spell Check Project" from the Extra menu to check all files in the current project for spelling errors. See Extra|Spell Check for a description of the spell check pane.

## Extra|Spell Check All

Extra|Spell Check All will activate the first file and start the spell check process. When the first file has been completed, the second file will be activated and the spell checker will be invoked on that file, until all files have been checked or you clicked the Abort button on the spell check page.

See Extra|Spell Check for a description of the spell check pane.

## Extra|Live Spelling

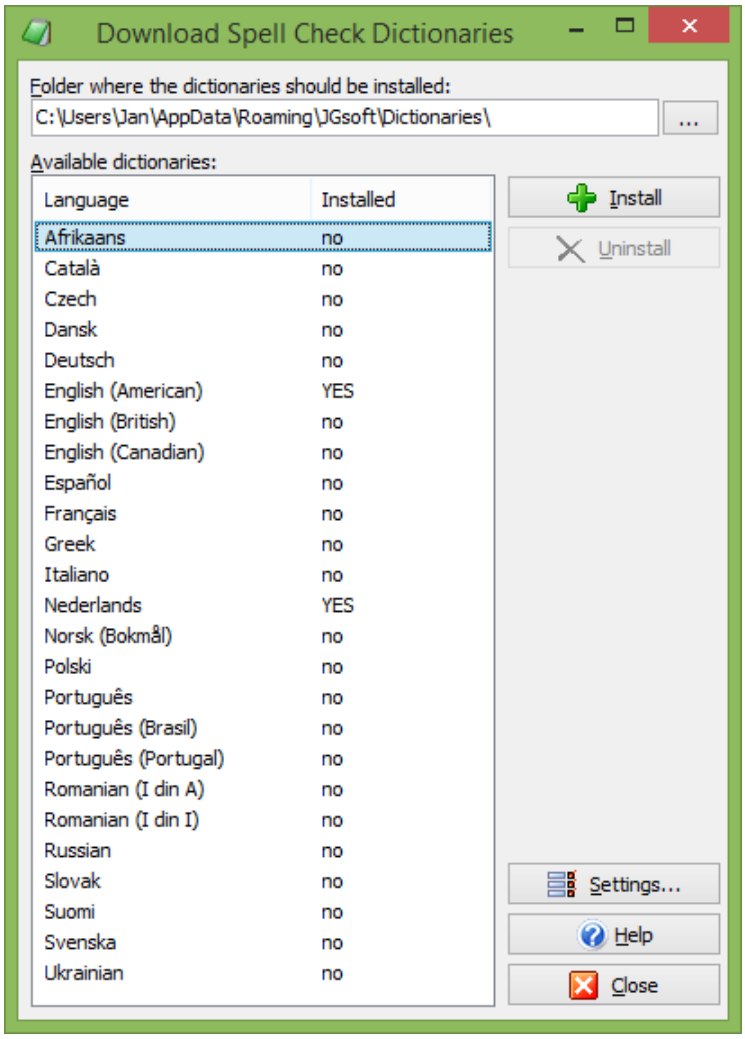
Pick Live Spelling from the Extra menu to turn on or off live spell checking for the current file. Live spell checking will mark all words that do not appear in the dictionary. By default, they will appear underlined and in red. You can change the look and whether live spelling is on or of by default in File Types|Colors.

Live spelling, as well as the regular spell check in EditPad Pro, works together with EditPad Pro's syntax coloring. Each syntax coloring scheme can exclude certain parts of the file from the spell check. Schemes for programming languages, such as the C++ and Java schemes, will restrict the spell checker to strings and comments. Those are likely to contain natural language, and therefore words that should appear in the dictionary. Other parts of the file are likely to contain specialized programming syntax, consisting of words and character sequences that are not English words.

If a word is marked as misspelled, you can double click on it to open the spell check pane. The pane will present you with a list of suggested replacements for the word, as well as allow you to ignore the word or add it to the dictionary. As soon as the word has passed the spell check, the spell check pane will close automatically.

## Download Spell Check Dictionaries

Before you can use Extra|Spell Check or Extra|Live Spelling, you need to download and install spell check dictionaries for one or more languages. You can easily do so right within EditPad Pro. Pick Options|Configure File Types from the menu, and click on the Colors and Syntax tab. Click the button "Download Spell Checker Dictionaries". EditPad Pro will then connect to the Internet to fetch a list of available languages.




If you want, you can choose the folder into which the dictionaries should be installed. If you specify a folder that does not exist, EditPad Pro will create it. All dictionaries must be installed into the same folder. If you already installed some dictionaries, and then install another dictionary into a different folder, EditPad Pro will move the previously installed dictionaries to the new folder.

To install a dictionary, click on the language you want in the list, and click the Install button. EditPad Pro will then automatically download the dictionary, and install it into the folder you specified. A progress meter will appear while downloading.

You can have as many dictionaries installed as you want. There is no need to uninstall dictionaries. However, if you want, you can select a language and click the Uninstall button to delete a spell check dictionary.

If you are behind a proxy server, and EditPad Pro is unable to detect your proxy settings, you can change them by clicking the Settings button.

Due to firewall and other network settings, it is possible that EditPad Pro cannot connect to the Internet directly. In that case, you can manually download and install the dictionaries from http://www.editpadpro.com/spell.html

## Extra|Sort Alphabetically A-Z

If no part of the active file has been selected, Extra|Sort Alphabetically will sort the entire file alphabetically, paragraph by paragraph.

To sort the entire file on a specific column of text, select that column of text before selecting Extra|Sort Alphabetically first. Make sure that the selection does not span more than one paragraph. All the paragraphs in the file will be sorted, using the selected column as the sort key.

If the selection spans more than one paragraph, then the selected paragraphs will be sorted. If you want to sort only a selected number of paragraphs on a certain column, make a rectangular selection first. The paragraphs that are partially covered by the rectangular selection will be sorted entirely as if the selection was a normal one. However, the sort order will not be determined by their entire paragraph text, but only by the text covered by the selection.

If you are working with a text files that contains information arranged in columns, padded with spaces, you could make a rectangular selection of the third column and then use Extra|Sort Alphabetically. The data will then be sorted on the third column.

The progress meter indicates the percentage of paragraphs that have already been sorted. The sort operation works from top to bottom. If you abort the operation, the paragraphs that have already been sorted will remain sorted, and the unsorted paragraphs will maintain their old position. If you want to cancel the sort altogether and revert its results, use Edit|Undo.

## Extra|Sort Alphabetically Z-A

Extra|Sort Alphabetically Z-A works just like Extra|Sort Alphabetically A-Z, except that the file or the selected text is sorted in reverse alphabetical order.

## Extra|Delete Duplicate Lines

Select the Delete Duplicate Lines item in the Extra menu to delete lines with (nearly) identical text on them. The status bar will indicate how many lines were deleted. You can make a number of choices as to what EditPad Pro will consider a duplicate line.

### Scope

If you've selected part of the file before using the Delete Duplicate Lines command, you can limit the command to delete only lines that are selected. If the first and/or last line in the selection are only partially selected, the selection will be expanded to include them entirely. If the selection is rectangular, lines covered by the selection will be deleted entirely.

### Proximity of Duplicate Lines

Select "anywhere in the scope" to delete all lines that are duplicated anywhere. The first copy of the line will remain, while all the others will be deleted. If you've set the scope to "selected lines", the lines must be duplicated inside the selection. Lines that are not duplicated inside the selection will not be deleted, even if they have lines outside the selection.

Select "adjacent lines only" if you only want to delete a line's duplicates if they're immediately below the line they duplicate, without any other lines between them. If the file's lines are sorted alphabetically, then the end result of "anywhere in the scope" and "adjacent lines only" will be the same. In a sorted file, all duplicates are sorted together. However, selecting "adjacent lines only" will delete the duplicate lines significantly faster, certainly when the number of lines in the file is large. If you select "anywhere in the scope", EditPad Pro has to compare each line with every other line in the file.

### Comparison Options

By turning on one or more comparison options, you can tell EditPad Pro to consider lines as duplicates even when they aren't identical.

The "compare selected columns only" is only available when you've made a selection that does not span more than one line, or when you've made a rectangular selection. With this option, EditPad Pro will only compare the selected columns. E.g. if the selection spans from column 10 to column 20, EditPad Pro will compare columns 10 through 20 of each line. If a line has less than 10 characters it will be considered blank. This has important consequences (see next section).

"Ignore differences in leading spaces and tabs" will treat lines that only differ in the number of spaces and tabs at the start of the line as duplicates. Similarly, "ignore trailing spaces and tabs" ignores differences in spaces and tabs at the end of each line. "Ignore all differences in spaces and tab" is more than a combination of the two previous options. EditPad Pro will then completely ignore all spaces and tabs, including spaces and tabs in the middle of lines.

"Ignore difference in case" compares lines without regard to the difference between upper case and lower case letters.

### Lines to Delete

You must select one or two choices in the "lines to delete" section. Every line in the file belongs to one of the 3 categories. Selecting none of the options would have no effect, and selecting all of them would delete all the lines in the file.

Turn on “2nd and following occurrences of duplicate lines” and turn off the other two options to delete all duplicate lines, leaving only unique files in the file, regardless of whether they were previously unique. Use this to delete unnecessary duplicates from a file.

Turn on both "2nd...“ and ”1st occurrence of duplicate lines" to delete all duplicate lines, leaving only lines that were previously unique.

Turn on both "2nd...“ and ”non-duplicated lines" to leave only one copy of all lines that had duplicates. If you paste the contents of two lists that consist of unique lines (when viewed separately) into a file in EditPad, then you can use this combination to get the lines that occurred in both files, but not the lines that occurred in only one of the files.

If you want to keep only lines that occur a certain number of times, use the Delete Duplicate Lines several times. E.g. if you only want lines that occur 3 times or more, use it twice with the "1st occurrence...“ and ”non-duplicated..." options turned on. Then use it again with the "2nd occurrence...“ and ”non-duplicated..." options. The first time you delete the lines that occur only once, the second time you delete lines that occur only twice, and the third time you delete the duplicates of lines that occur four times or more.

### Blank Lines

Since blank lines are technically all duplicates of each other, EditPad Pro offers you an extra choice for blank lines. You can choose to either delete all blank lines, not to delete any blank lines, or to only delete duplicate blank lines. The “duplicate blank lines” option takes into account the “proximity” setting, deleting either all but the first blank lines (“anywhere in the scope”), or only replacing subsequent blank lines with a single blank line (“adjacent lines only”).

If you’ve turned on the “compare selected columns only” option, a line may be considered blank even when it isn’t. If a line is shorter than the leftmost column in the selection, it is considered to be blank, even if it does have text on it.

Lines with only spaces and tabs on them are only considered to be blank if you’ve turned one of the options to ignore differences in spaces and tabs. On a line with only spaces and tabs, all spaces and tabs are considered to be both leading and trailing at the same time.

## Extra|Delete Blank Lines

Select Delete Blank Lines in the Extra menu to delete all totally blank lines from the file. Only lines that have no characters at all are deleted. Lines that consist solely of whitespace are not deleted.

If you want to delete lines that have whitespace too, first use Extra|Trim Trailing Whitespace to remove the whitespace, and then use Delete Blank Lines.

## Extra|Consolidate Blank Lines

Select Delete Blank Lines in the Extra menu to replace all blocks of consecutive totally blank lines with a single blank line. Only lines that have no characters at all are consolidated. Lines that consist solely of whitespace are ignored.

If you want to consolidate lines that have whitespace too, first use Extra|Trim Trailing Whitespace to remove the whitespace, and then use Consolidate Blank Lines.

## Extra|Trim Leading Whitespace

Select Trim Leading Whitespace in the Extra menu to remove whitespace from the start of each line. If you selected part of the file, only lines that area selected will be trimmed. If not, all lines in the file will be trimmed.

If you've made a rectangular selection, EditPad Pro will only delete whitespace inside the rectangular block. Whitespace in the leftmost selected column and adjacent columns to the right will be deleted. This whitespace needn't be at the start of the line

## Extra|Trim Trailing Whitespace

Select Trim Trailing Whitespace in the Extra menu to remove whitespace from the end of each line. If you selected part of the file, only lines that area selected will be trimmed. If not, all lines in the file will be trimmed.

If you've made a rectangular selection, EditPad Pro will only delete whitespace inside the rectangular block. Whitespace in the rightmost selected column and adjacent columns to the left will be deleted. This whitespace needn't be at the end of the line. If a line is short and doesn't extend to the rightmost selected column, any whitespace at the end of the line covered by the selection will be deleted.

## Extra|Trim Whitespace

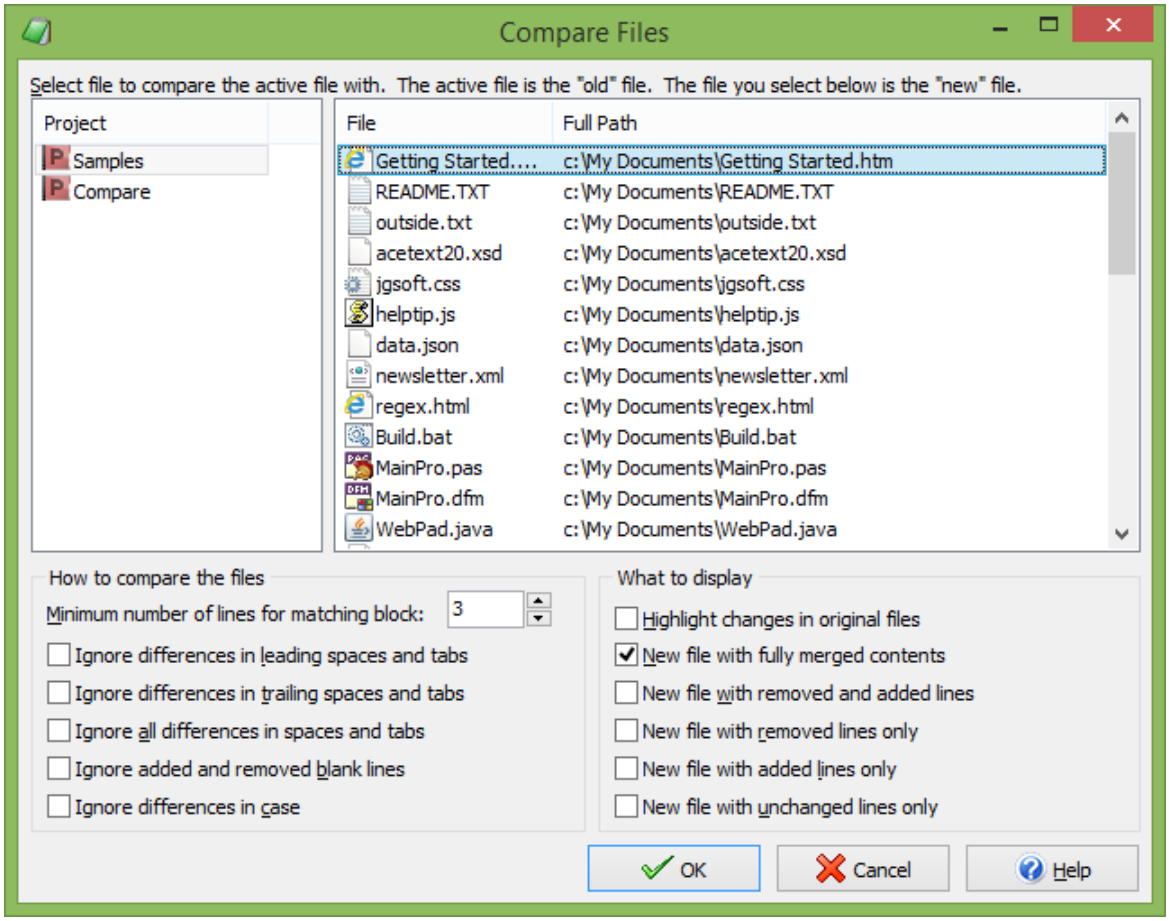
Select Trim Whitespace in the Extra menu to remove whitespace from the start and the end of each line. If you selected part of the file, only lines that area selected will be trimmed. If not, all lines in the file will be trimmed.

If you've made a rectangular selection, EditPad Pro will only delete whitespace inside the rectangular block. Whitespace in the leftmost selected column and adjacent columns to the right will be deleted. Whitespace in the rightmost selected column and adjacent columns to the left will also be deleted.

## Extra|Compare Files

If you have two versions of the same text file, you can use Extra|Compare Files to visualize the differences between those files. After that, you can edit the generated difference file to merge both versions into a single, new file.

First you need to open the two files you want to compare. The Extra|Compare Files menu item will be grayed out until you have two or more files open.


Compare Files
Select file to compare the active file with. The active file is the "old" file. The file you select below is the "new" file.
Project
Samples
Compare
File
Full Path
Getting Started.... c:\My Documents\Getting Started.htm
README.TXT c:\My Documents\README.TXT
outside.txt c:\My Documents\outside.txt
acetext20.xsd c:\My Documents\acetext20.xsd
jgsoft.css c:\My Documents\jgsoft.css
helptip.js c:\My Documents\helptip.js
data.json c:\My Documents\data.json
newsletter.xml c:\My Documents\newsletter.xml
regex.html c:\My Documents\regex.html
Build.bat c:\My Documents\Build.bat
MainPro.pas c:\My Documents\MainPro.pas
MainPro.dfm c:\My Documents\MainPro.dfm
WebPad.java c:\My Documents\WebPad.java
How to compare the files
Minimum number of lines for matching block: 3
Ignore differences in leading spaces and tabs
Ignore differences in trailing spaces and tabs
Ignore all differences in spaces and tabs
Ignore added and removed blank lines
Ignore differences in case
What to display
Highlight changes in original files
New file with fully merged contents
New file with removed and added lines
New file with removed lines only
New file with added lines only
New file with unchanged lines only
OK
Cancel
Help

### Selecting Files

To compare the files, first activate the file that contains the original or older version of the document by clicking on its tab. Then select Extra|Compare Files from the menu. In the window that appears, click on the file that contains the newer version of the document, in the list of files that are currently open. The active file is omitted from that list, since you cannot compare a file with itself.

The file that was active at the time you selected Extra|Compare Files in the menu will be considered the "old" file. The file that you select in the file comparison options screen will be considered the "new" file.

## How to Compare The Files

You can specify several options when comparing files. The setting for the minimum match size will be explained in detail later. 3 is a good default value.

You can choose to ignore differences in leading spaces and tabs. That is, when comparing two lines, EditPad Pro will ignore any space and tab characters at the start of the lines. Similarly, you can choose to ignore trailing spaces and tabs. That is, spaces and tabs at the end of a line. You can mark both options to ignore both leading and trailing spaces and tabs. If you mark “ignore all differences in spaces and tabs”, then EditPad Pro will ignore any spaces and tabs throughout both files when comparing their lines. These options are useful when comparing files where differences in whitespace have little or no meaning.

If you turn on “ignore added and removed blank lines”, EditPad Pro will not add blank lines that occur only in one of the two compared files to any new file created by the file comparison. Those lines will also not be highlighted in the original files. Lines with only spaces and tabs will be considered to be blank only if you’ve turned on one of the options to ignore differences in spaces and tabs.

Finally, “ignore differences in case” tells EditPad Pro to ignore whether a character is uppercase or lowercase, making “A” identical to “a”.

## What to Display

If you select to highlight changes in the original files, EditPad Pro will mark lines present in the “old” or “original” file in red when they are not present in the “new” or “edited” file. It will also mark lines in the “new” file in green when they are not present in the “old” file. You can configure the "compare files: deleted line“ and ”compare files: added line" colors in the Colors Preferences.

If you turn on “new file with fully merged contents”, EditPad Pro will create a new tab with the two files merged, based on their differences. The tab will be labeled “OldFile compared with NewFile”. Lines that are identical in both files are added once and displayed on a normal background. Lines only present in the “old” file are added on a red background, and lines only present in the “new” file are added on a green background. If the text on a line was changed between the “old” and “new” versions of the file, EditPad Pro will first add the old version of the line on a red background, and then the new version of the line on a green background. If consecutive lines where changed, EditPad Pro will first add a block with the old versions of all the consecutive lines, and then the new versions of the consecutive lines.

“New file with removed and added lines” is the same as “new file with fully merged contents”, except that lines present in both the “old” and “new” files are not added. Lines are highlighted red and green. Consecutive changed lines are grouped in the same way. The tab will be labeled “Differences between OldFile and NewFile”.

“New file with removed lines only” contains a tab with lines that are present in the “old” file, but not in the “new” file. Since all lines are old lines, none of them are highlighted. The tabs label will be “Lines from OldFile removed in NewFile”.

Similarly, “new files with added lines only” creates a tab labeled “Lines not in OldFile added to NewFile” with lines present in the “new” file, but not in the “old” file.

Finally, "new file with unchanged lines only" creates a tab labeled "Lines unchanged between OldFile and NewFile". This tab will show all the lines present in both files.

All new tabs with difference output behave like a regular files in EditPad Pro. You can use any editing command on them and save them for later use. If you open the View|File History, it will remind you of the two files that were compared, and which was the "old" or "original" file, and which was the "new" or "edited" file.

## Minimum number of lines for matching block

When you pick Extra|Compare Files from the menu, EditPad Pro will ask for a file to compare the active one with. Near the bottom of the selection window, you will see the spinner box labeled "minimum number of lines for a matching block".

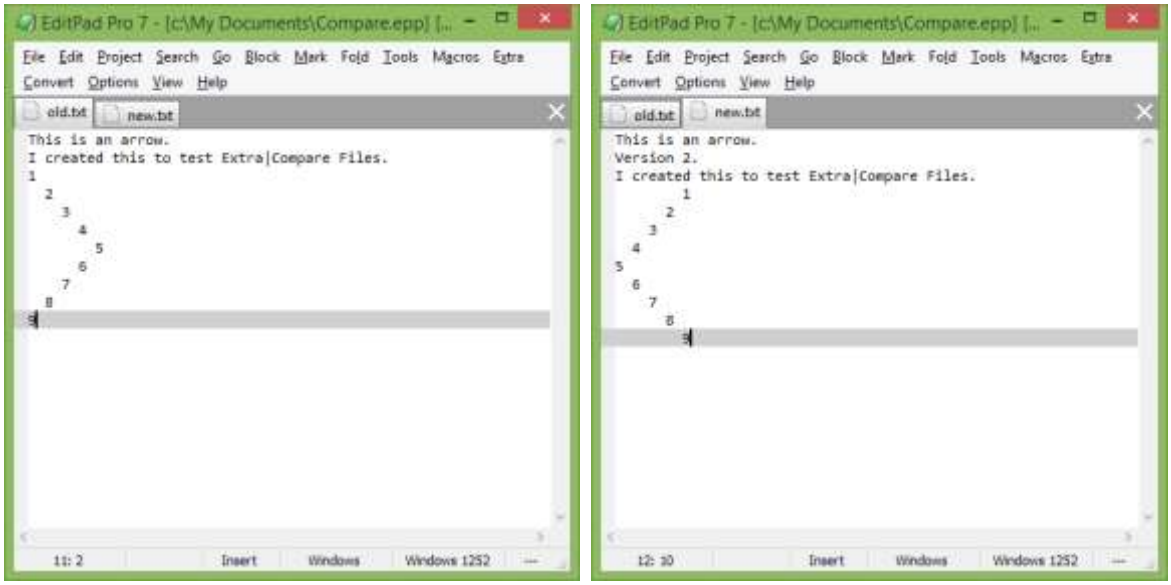
The default setting is 3. You can set it to any number from 1 up to and including 9.

This value is used by EditPad Pro in the following situation. While comparing the files, it encounters a section of one or more changed paragraphs. In the difference output, these paragraphs are grouped into a single red block showing the original paragraphs, followed by a single green block with the new paragraphs. This is done because all those paragraphs are most likely to be connected somehow: a chapter in a book that was heavily edited, a source code routine that has been modified, etc. Keeping those paragraphs together maintains their logical connection so you can easily see what happened when inspecting the output of Extra|Compare Files.

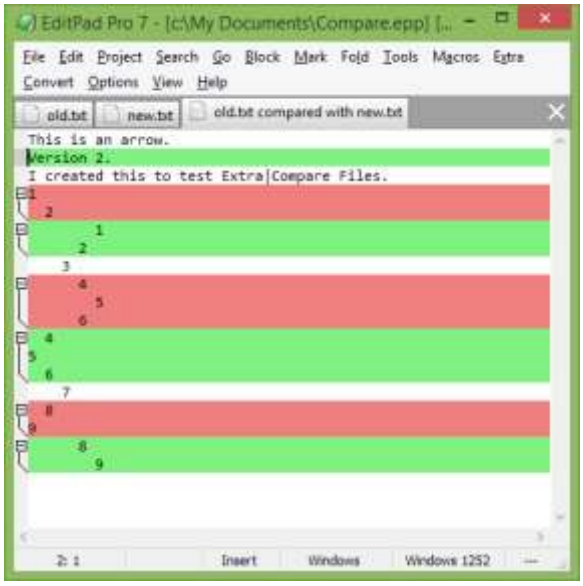
To even better maintain the logical structure of the document, you can have EditPad Pro also include a few paragraphs in the block that are identical in both versions of the document. This will happen when you set "minimum number of lines for a matching block" to a number greater than one (1).

The setting indicates the minimum number of consecutive paragraphs that are identical in both files are required for EditPad Pro to stop grouping the modified paragraphs and add the identical paragraphs as an unchanged block. The graphical illustrations below make this clear.

First you see the old and new version of our test document:




Then we use Options|Compare Files, and set "minimum number of lines for a matching block" to one, effectively turning off the feature. This is the result:




You see that EditPad Pro detects the line "Version 2" as being inserted.
It also detects that the lines numbered 3 and 7 are present in both files, and that the blocks 1-2, 4-6 and 8-9 have changed. This is technically exact (it is the optimum solution), but logically it makes no sense. The arrow is broken into pieces.

If we try again and set "minimum number of lines for a matching block" to anything greater than 1, we get the following result:

EditPad Pro 7 - [c:\My Documents\Compare.epp] [...
File Edit Project Search Go Block Mark Fold Tools Macros Extra
Convert Options View Help
old.txt compared with new.txt
old.txt compared with new.txt
This is an arrow.
Version 2.
I created this to test Extra|Compare Files.
Insert
Windows
Windows 1252

Much better! The "I created..." line is still included on its own since it does not follow a block of changed paragraphs, but an inserted one.

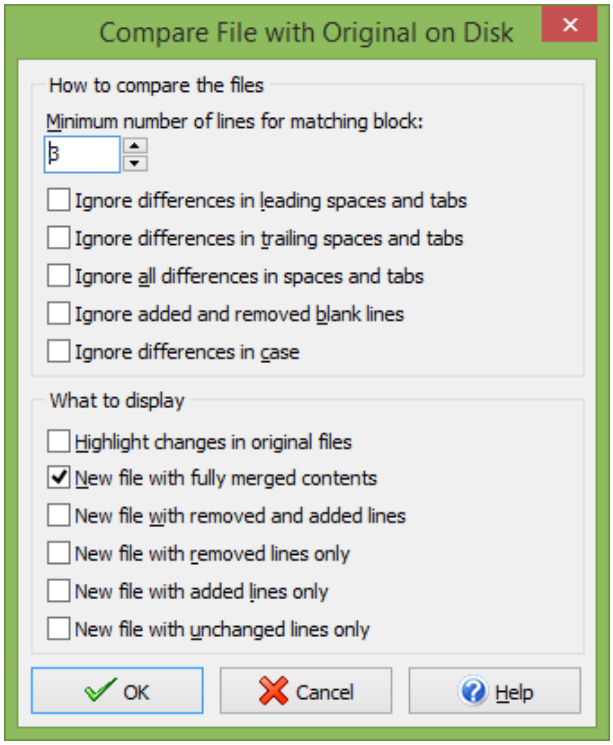
However, the lines containing the numbers 3 and 7, even though they are present in both files, are displayed in the middle of the changed block as if they had been changed too. This way, our logical structure, the arrow, is kept together nicely and we can instantly see what happened: the arrow's direction was changed. This was not at all so clear in our first comparison attempt.

Since you will not be comparing arrows but real documents such as source code files, the "minimum number of lines for a matching block" can be configured for the task at hand. Many source code files contain lines with nothing but a curly brace or a "begin" or "end". These lines add structure to the source, and not content. When comparing two versions of a piece of source, these lines are likely to be matched all over the place, breaking up the logical structure of the source code in the difference output.

If things don't look right, experiment with the "minimum number of lines for a matching block" setting.

## Extra|Compare with File on Disk

The Compare with File on Disk item in the Extra menu works just like the Compare Files item, with one key difference. Whereas Compare Files asks you which file you want to compare the active file with, Compare with File on Disk compares the active file with the same file on disk. If you have made changes to a file in EditPad Pro and you haven't saved those changes yet, you can use Compare with File on Disk to see the changes that you made.


Compare File with Original on Disk
How to compare the files
Minimum number of lines for matching block:
3
Ignore differences in leading spaces and tabs
Ignore differences in trailing spaces and tabs
Ignore all differences in spaces and tabs
Ignore added and removed blank lines
Ignore differences in case
What to display
Highlight changes in original files
New file with fully merged contents
New file with removed and added lines
New file with removed lines only
New file with added lines only
New file with unchanged lines only
OK
Cancel
Help

## Extra|Next Comparison Mark

If you have used Extra|Compare Files or Extra|Compare with File on Disk, you can use Extra|Next Comparison Mark to find the block of differing paragraphs after the text cursor's position.

## Extra|Previous Comparison Mark

If you have used Extra|Compare Files or Extra|Compare with File on Disk, you can use Extra|Previous Comparison Mark to find the block of differing paragraphs before the text cursor's position.

## Extra|Clear Comparison Marks

After you've used Extra|Compare Files or Extra|Compare with File on Disk, the files you compared may have their differences highlighted. If you're done comparing the files but want to continue working with them in EditPad Pro, select Clear Comparison Marks in the Extra menu to remove the red and green highlights.

If you don't want to continue working with the files, simply close them. You don't need to remove the comparison marks first.

## Extra|Statistics

Choose Statistics from the Extra menu to have EditPad Pro compute a bunch of statistics about the file you are currently editing. EditPad Pro will create a new tab to hold the statistics. An example:

```
                            Filename:
S:\JGsoft\EditPad7\Help\source\extrastatistics.txt
                          Last saved: 11-Feb-2011 11:32
                           File size: 3,260 bytes
               Number of paragraphs:    41
                    Number of words:   497
      Number of letters and digits: 2,332
    Number of printable characters: 2,594
        Total number of characters: 3,260
                     Selection size:   638 bytes
                     Selected words:    69
       Selected letters and digits:   338
     Selected printable characters:   365
Total number of selected characters:   638
```

The first two lines indicate under which name the file was saved, and when it was last saved. The third line indicates the size of the file in bytes. The total file size is the total number of characters in the file, including hidden character such as line breaks.

If part of the file was selected before you picked Extra|Statistics from the menu, then the size in bytes of the selection is indicated as well.

"Number of paragraphs" indicates the number of paragraphs in the file. A paragraph is a sequence of characters terminated by pressing Enter on the keyboard, which inserts a hard return into the file. If you turn off word wrapping, then each paragraph is displayed as one line. The number of lines is not indicated in the statistics, because it varies with word wrapping.

Number of words indicates the number of sequences of "letters and digits", separated by sequences of "non-letters". A sequence consisting of a single letter or digit is also considered a word. A "letter or digit" is any character that is matched by the character class «\w» in regular expressions. This includes letters, digits, and underscores in all scripts. You can test what EditPad Pro considers a letter by picking Search|Prepare to Search from the menu, typing «\w» into the search box, turning on the option "regular expressions", and clicking the Search button. To search for words, use \w+ (slash lowercase w plus) instead.

"Number of characters" indicates the number of printable characters. This includes all characters except spaces, tabs, line breaks and other control characters. "Total number of characters" counts all characters, including whitespace, line breaks, and control characters.

## Extra|Project Statistics

Like Extra|Statistics, Project Statistics will create a new tab with statistics. It will compute the same statistics, but for all files in the current project rather than the current file only. The order of the files in the statistics is the same order as in which they are arranged in EditPad Pro's tabs. If you want the files to be sorted alphabetically, use View|Sort Tabs Alphabetically prior to using Extra|Statistics for All Files.

## Extra|Statistics for All Files

Like Extra|Statistics, Statistics for All Files will create a new tab with statistics. It will compute the same statistics, but for all files in all projects rather than the current file only. The order of the files in the statistics is the same order as in which they are arranged in EditPad Pro's tabs. If you want the files to be sorted alphabetically, use View|Sort Tabs Alphabetically prior to using Extra|Statistics for All Files.

# 12. Convert Menu

## Convert|To Uppercase

Converts the current selection to all uppercase letters. If there is no selection, the line the cursor is on is converted to uppercase.

## Convert|To Lowercase

Converts the current selection to all lowercase letters. If there is no selection, the line the cursor is on is converted to lowercase.

## Convert|Initial Caps

Turns all the letters in the current selection into lowercase letters, except for the first letter of each word which becomes uppercase. If there is no selection, the line the cursor is on is converted to initial caps.

## Convert|Invert Case

Turns all uppercase letters in the selection into lowercase letters, and all lowercase letters into uppercase. If there is no selection, the line the cursor is on is inverted.

If you have been typing with Caps Lock on, you do not have to start over. Let this function come to the rescue.

## Convert|Text Encoding

Computers deal with numbers, not with characters. When you save a text file, each character is mapped to a number, and the numbers are stored on disk. When you open a text file, the numbers are read and mapped back to characters. When saving a file in one application, and opening the that file in another application, both applications need to use the same character mappings.

Traditional character mappings or code pages use only 8 bits per character. This means that only 256 characters can be represented in any text file. As a result, different character mappings are used for different language and scripts. Since different computer manufacturers had different ideas about how to create character mappings, there's a wide variety of legacy character mappings. EditPad supports a wide range of these.

In addition to conversion problems, the main problem with using traditional character mappings is that it is impossible to create text files written in multiple languages using multiple scripts. You can't mix Chinese, Russian and French in a text file, unless you use Unicode. Unicode is a standard that aims to encompass all traditional character mappings, and all scripts used by current and historical human languages.

## How to Make a File Readable in EditPad

If you've received a text file from another person, or opened a file created on another computer, it may not immediately be readable in EditPad. Two things need to be set right. First, you must use a font that contains the characters you want to display. While all fonts contain English characters, far fewer fonts contain Chinese, Thai, or Arabic characters. In EditPad, select Options|Font in the menu. Make sure to select the script you want in the lower right corner of the font selection screen.

If you use multiple scripts in a single file, then you probably won't have a single font that can (nicely) display all of those scripts. Fortunately, EditPad can use any number of fonts at the same time, and automatically use each font only for those scripts that it supports. Select Options|Text Layout in the menu. Set "text layout and direction" to "complex script". Select a main font and any number of fallback fonts. If the main font doesn't support certain script, EditPad displays that script using the first fallback font that does support it.

If the file still isn't displayed property after you've selected a proper font, the file was saved with a different character encoding than EditPad is using. Select Text Encoding in the Convert menu to change the encoding EditPad uses for that file.

At the top of the screen, you will see part of the file as EditPad interprets it now, along with the encoding used. Make sure the **"interpret the original data as being encoded with another character set"** is marked. Then, try selecting a new encoding. The result will appear immediately in the preview at the bottom of the screen. Keep trying different encodings until you find one that produces a readable file.

If the file was created on a computer running Windows, try the Windows encodings first. All Windows computers use one of the Windows encodings as the default. UTF-8 and UTF-16 little endian are also likely, as Unicode is becoming popular among modern Windows applications.

If the file was created on a computer running a UNIX variant such as Linux, try the ISO-8859 and EUC encodings. UTF-8 and UTF-16 little endian are also likely.

If the file was created on a modern Mac running OS X, it probably uses UTF-8. If it was created on an older Mac, try the Mac encodings.

If the file was created by a old DOS application, try one the DOS character sets. The DOS character sets were used by Microsoft's MS-DOS and DOS versions from other companies. If you know that the file is supposed to contain "line drawing symbols", the DOS character sets are also very likely. The DOS character sets are the only ones that contain line drawing symbols. DOS applications used characters that look like lines, bevels and corners to draw pseudo-graphical interfaces on character-based screens. DOS predates Unicode, so the Unicode formats are unlikely, even if they contain line drawing symbols.

If the file is written in Russian or Ukrainian, the KOI8-R and KOI8-U encodings are very likely candidates, even for files created on Windows or UNIX systems. Particularly ISO-8859-5 has never reached the popularity of KOI8.

If the file was created on an old mainframe or an IBM AS/400 (renamed iSeries) system, try the EBCDIC encodings. EBCDIC was the de facto standard in the days computers used punch cards. If EBCDIC doesn't produce a readable file, try the DOS character sets, which were used by IBM's PC-DOS.

## Non-Representable Characters Replaced with Question Marks

There are two ways in which a text editor can keep files in memory while editing them. Some editors use Unicode internally. On Windows that is typically UTF-16 LE. When you open a file that uses any other encoding, it is converted into UTF-16 LE in memory. When you save the file, it is converted back into the other encoding. The benefit is that the developers of such editors have only one encoding to deal with for all editing functions. The downside is that the conversion takes extra time and extra memory. Such editors usually don't perform with very large files. If a file is loaded with the wrong encoding, it has to be reloaded with the correct one. If you don't notice the wrong encoding is being used, or if it contains bytes that are invalid for the encoding that the file actually uses, data loss may occur. There's no way to preserve invalid byte sequences when converting to UTF-16.

When EditPad loads a file, it loads its actual bytes into memory. In EditPad Pro you can even see those bytes by switching to hexadecimal mode. That is what the term "original data" in the choices in the Convert|Text Encoding window refers to. Because EditPad keeps the file's original bytes in memory, it can instantly change how it interprets those bytes. Simply use the "interpret" option in the Text Encoding window and select another encoding. This option does not change the contents of the file at all, nor is there any need to reload the file. It only changes how EditPad translates the bytes into characters.

While you edit a file, EditPad converts those bytes into characters on-the-fly for display. When you type in or paste in new text, EditPad immediately converts the characters you type into the appropriate bytes for the file's encoding. That means if you're file doesn't use Unicode, you can only type in or paste in characters that are supported by the file's encoding. EditPad's Character Map can show you all those characters.

If you paste in characters that are not supported by the file's encoding, EditPad has no way to convert those characters into bytes to store them into the file. Such characters will be lost. They are permanently changed into **question marks** to indicate the actual characters you tried to paste could not be represented. In order to paste the actual characters, first use Edit|Undo to remove the question marks. Then use Convert|Text Encoding with the "encode original data with another character set" option. Select a Unicode transformation or any encoding that supports the characters you want to paste, as well as those already present in the file. EditPad then changes the bytes in the file to represent the same characters in the new encoding. Now you can paste your text again and get the actual characters. Note that you have to undo pasting the question marks. Changing the encoding does not magically restore the characters. Since EditPad Pro uses the file's actual encoding for in-memory storage rather than Unicode, newly typed or pasted characters that cannot be represented cannot be stored.

## How to Make an EditPad File Readable by Others

To make sure other people can read files you've written in EditPad, simply save it in an encoding that the other person can read. If he or she also uses Windows, you don't need to do anything. EditPad's default text encoding settings save the file in your computer's default Windows code page.

If the file is written in English, you also have little to worry about. English text is encoded the same way in UTF-8, all Windows code pages, all ISO-8859 code pages, all DOS code pages, all Mac code pages, and also KOI8.

If your document uses non-English characters, and you're not sure which encodings the other person can read, the UTF-8 encoding is a safe bet. UTF-8 files start with a byte order marker that identify them. EditPad and many other applications running on Windows, Linux and Mac OS X detect the byte order marker, and automatically interpret the file as UTF-8. Since UTF-8 is a Unicode transformation, it supports all modern

human languages. All characters present in any of the non-Unicode code pages supported by EditPad are also present in the Unicode mapping.

To change a file's encoding, select Text Encoding in the Convert menu. Mark the **"encode original data with another character set"** option and select the encoding you want to convert the file into. If you get a bold red warning that some characters could not be converted, this means that the encoding you are trying to convert the file into cannot represent some characters you've used in the file. Those characters will be replaced by question marks if you proceed with the conversion. The Unicode encodings are the only ones that can represent all characters from all human languages. The others are typically limited to English plus one language (e.g. Chinese) or one group of languages (e.g. Western European languages).

## Convert|To Windows (CR LF)

Converts the active file to the line break style used by Windows and DOS text files. These operating systems terminate each line with a Carriage Return and Line Feed pair. Only the line termination characters are converted. Anything else remains untouched.

To EditPad, it does not matter whether a file uses Windows, UNIX, or Mac line breaks. EditPad even handles files that use a mixture of all three. However, other applications often only support the "proper" line breaks for their environment. Windows applications like Notepad display UNIX text files with everything on one line.

Use the Convert|To Windows command to make sure your file uses consistent Windows line breaks if you'll be using the file with other Windows applications. If the Convert|To Windows menu item is grayed out, that means the file already uses Windows line breaks consistently.

If line break styles matter for your work, use the Status Bar Preferences to enable a status bar indicator to show the file's (dominant) line break style. In EditPad Pro, in the Editor Preferences you can configure Options|Visualize Line Breaks to display the actual line break style for each line rather than a generic paragraph marker.

## Convert|To UNIX (LF only)

Converts the active file to the line break style used by UNIX and derivatives like Linux, BSD, and even OS X. These operating systems terminate each line with a single Line Feed character. Only the line termination characters are converted. Anything else remains untouched.

To EditPad, it does not matter whether a file uses Windows, UNIX, or Mac line breaks. EditPad even handles files that use a mixture of all three. However, other applications often only support the "proper" line breaks for their environment. UNIX applications often try to display the CR in Windows text files, resulting in a weird character at the end of each line.

Use the Convert|To UNIX command to make sure your file uses consistent UNIX line breaks if you'll be using the file with other UNIX, Linux, or OS X applications. If the Convert|To UNIX menu item is grayed out, that means the file already uses UNIX line breaks consistently.

If line break styles matter for your work, use the Status Bar Preferences to enable a status bar indicator to show the file's (dominant) line break style. In EditPad Pro, in the Editor Preferences you can configure Options|Visualize Line Breaks to display the actual line break style for each line rather than a generic paragraph marker.

## Convert|To Macintosh (CR only)

Converts the active file to the line break style used by the Apple Macintosh. On the Macintosh, text files use a single Carriage Return character to terminate a line. Note that OS X applications may use the LF only UNIX format rather than the classic Mac CR only format. Only the line termination characters are converted. Anything else remains untouched.

To EditPad, it does not matter whether a file uses Windows, UNIX, or Mac line breaks. EditPad even handles files that use a mixture of all three. However, other applications often only support the "proper" line breaks for their environment. Classic Mac applications may try to display the LF in Windows text files, resulting in a weird character at the start of each line.

Use the Convert|To Macintosh command to make sure your file uses consistent classic Mac line breaks if you'll be using the file with older Mac applications. If the Convert|To Macintosh menu item is grayed out, that means the file already uses classic Mac line breaks consistently.

If line break styles matter for your work, use the Status Bar Preferences to enable a status bar indicator to show the file's (dominant) line break style. In EditPad Pro, in the Editor Preferences you can configure Options|Visualize Line Breaks to display the actual line break style for each line rather than a generic paragraph marker.

## Convert|Wrapping -> Line Breaks

When Options|Word Wrap is on, EditPad will break up long lines to they fit within the width of the EditPad window or are no longer than a certain number of characters. This wrapping is dynamic. It is not saved into the file. Therefore, the file will look different if you change the word wrap settings or open it in another application.

Use Convert|Wrapping -> Line Breaks if you want to solidify a text in its wrapped state. This will convert all soft line breaks introduced by the wrapping into hard line breaks as if you had pressed Enter at each position where a long line is wrapped.

You will also be asked if you want to align the text at the left side only, or if you want to align it at both sides. If you choose to align to the left side only, only line breaks will be inserted, as explained above. If you choose to align to both sides, EditPad Pro will also inserts spaces into each line, so that each line has the same number of characters. The spaces will be distributed evenly across each line. They will be inserted next to spaces already present on the original line. When the file is viewed with a fixed-width font such as Courier New, both the left and the right side of the text will be perfectly aligned.

This function will also turn off word wrap.

If you want to limit the maximum length of each line in the file, you can first set the maximum line length by using the drop-down menu of the word wrap button on the toolbar. Then use Convert|Wrapping->Line Breaks.

You should use Convert|Wrapping -> Lines when you have finished editing a file that will be used by an application or system that does not support word wrapping, or does not support lines longer than a certain number of character. For example, when preparing a message to be posted to a Usenet newsgroup or a message to be sent via email, you should set word wrap to 72 characters and then use Convert|Wrapping -> Lines. There are still a lot of newsreaders and email clients in use that cannot word wrap messages.

## Convert|Line Breaks -> Wrapping

Convert|Line Breaks -> Wrapping attempts to do the opposite of Convert|Wrapping -> Line Breaks. Email clients and newsreaders, for example, will often limit the maximum line length, inserting additional line breaks into the file that you did not type in by pressing Enter onto the keyboard. This is done because a lot of email and newsgroup software, mostly older software, still cannot handle long lines.

The problem with this is that after the hard line breaks have been inserted, the text becomes difficult to edit. The text will not be nicely and automatically rewrapped when you insert new text into a middle of the paragraph, like EditPad does when word wrap is on. With Convert|Line Breaks -> Wrapping you can tell EditPad Pro to attempt to remove the line breaks that were inserted into the file for the purpose of emulating word wrap. I did say "attempt", because EditPad Pro has no way of detecting whether a line break was automatically inserted to limit the length of the line, or if it was inserted by a person pressing Enter on the keyboard. Technically, there is no difference between the two. But usually, EditPad Pro does a remarkably good job.

## Convert|Double -> Single Spacing

A text file is double-spaced if there is a blank line between each line of text. Select Double -> Single Spacing from the Convert menu to remove those blank lines. If you select multiple lines of text first, only the the selected block is converted. If there is no selection or the selection does not span multiple lines, then this conversion affects the entire file.

If you use the double to single spacing conversion on a file that is not double-spaced, EditPad Pro will remove as many alternating blank lines as possible. If part of the file is single spaced and part is double spaced, the whole file will become single spaced. If the entire file is single spaced, but blank lines are used to separate paragraphs, then those blank lines will be removed.

Certain files will appear double-spaced in EditPad, and single-spaced in other applications such as Notepad. This is caused by a different interpretation of line break characters. Usually, those files were incorrectly transferred between a Windows and UNIX system, such as a PC and a web server. This causes incorrect line break characters to be added to the file. You can easily fix such files in EditPad Pro. First, select Convert|Double -> Single Spacing from the menu to remove the unwanted blank lines. Blank lines that appeared in the original file will remain. Then, select Convert|To Windows to make all the line breaks consistent with the Windows format. If Convert|To Windows is grayed out (i.e. unavailable), then the file already uses the correct line breaks.

## Convert|Single -> Double Spacing

A text file is double-spaced if there is a blank line between each line of text. Select Single -> Double Spacing from the Convert menu to insert a blank line between each line of text, making the current file double spaced. If you select multiple lines of text first, only the the selected block is converted. If there is no selection or the selection does not span multiple lines, then this conversion affects the entire file.

Note that EditPad Pro will not force you to maintain the double spaced format of the file. Pressing Enter will insert a single line break as usual. Press Enter twice to insert a double line break. You can type text into the blank lines added by the conversion if you want. This could be useful to annotate a manuscript, for example.

## Convert|Tabs -> Spaces

Converts all tab characters in the active file into spaces. This command will only have its desired effect when using a fixed-width font like Courier New. If you select multiple lines of text first, only the the selected block is converted. If there is no selection or the selection does not span multiple lines, then this conversion affects the entire file.

The number of spaces used for each tab by the conversion is the same the number of spaces used for each tab to display the file. It can be set in File Types|Editor.

## Convert|Spaces -> Tabs

Converts as many spaces into tab characters as possible, without changing the layout of the file. This function will only have its desired effect when using a fixed-width font like Courier New. If you select multiple lines of text first, only the the selected block is converted. If there is no selection or the selection does not span multiple lines, then this conversion affects the entire file.

The number of spaces used for each tab by the conversion is the same the number of spaces used for each tab to display the file. It can be set in File Types|Editor.

## Convert|Indentation Spaces -> Tabs

Converts as many spaces at the start of each line into tab characters as possible, without changing the layout of the file. Spaces that occur after a non-whitespace character on a line are not converted. This function will only have its desired effect when using a fixed-width font like Courier New. If you select multiple lines of text first, only the the selected block is converted. If there is no selection or the selection does not span multiple lines, then this conversion affects the entire file.

The number of spaces used for each tab by the conversion is the same the number of spaces used for each tab to display the file. It can be set in File Types|Editor.

## Convert|Characters -> \uFFFF

Select the Characters -> \uFFFF command in the Convert menu to replace all non-ASCII characters in the file with an escape sequence in the form of `\uFFFF`. If you select some text before using this command, only non-ASCII characters in the selection are replaced. Otherwise, all non-ASCII characters in the file are replaced.

Using this command does not change the encoding EditPad Pro uses for this file. Saving the file after replacing all non-ASCII characters this way only results in an ASCII file if the encoding you're using for the file is compatible with ASCII. For example, the Windows code pages will result in an ASCII file, but UTF-16 will not.

In Convert|Text Encoding you can select "ASCII + \uFFFF Unicode Escapes" as the file's encoding. When using this encoding, EditPad Pro displays the actual characters for all `\uFFFF` escapes in the file. When you type non-ASCII characters into the file, EditPad Pro will display the character you typed, but store its `\uFFFF` escape in the file. If you type "après-ski" using this encoding, EditPad Pro will display "après-ski", but store "apr\u00E8s-ski" in the file. If you open that file in Notepad (which cannot interpret Unicode escapes), you'll see "apr\u00E8s-ski".

Use the Convert|Characters -> \uFFFF along with an encoding such as UTF-8 or Windows 1252 when you want to use Unicode escapes in only part of the file. EditPad will display the Unicode escapes while you edit the file. You can see where Unicode escapes are used and where actual characters are used.

Use the "ASCII + \uFFFF Unicode Escapes" encoding when working with files where Unicode escapes are the only way of representing non-ASCII characters. EditPad will display the actual characters while you edit the file. You won't see the Unicode escapes. EditPad makes sure all newly typed or pasted characters are stored as Unicode escapes in the file.

If you have a file that contains non-ASCII characters that are not written as Unicode escapes then you can convert that file you being pure ASCII + \uFFFF Unicode Escapes in three steps. First, open the file in EditPad and make sure all characters are displayed correctly. If not, use Convert|Text Encoding to select the proper encoding with the "interpret" option. Second, use Convert|Characters -> \uFFFF to convert all non-ASCII characters to Unicode escapes. The actual escapes will appear. Finally, use Convert|Text Encoding and select to convert to ASCII + \uFFFF Unicode Escapes. The Unicode escapes now again appear as actual characters. If you save the file the, it will be saved as pure ASCII with Unicode escapes.

In the Encoding section in the file type configuration, there is an option to make EditPad Pro detect whether a file consists of pure ASCII with Unicode Escapes. If that option is on, EditPad Pro automatically selects the "ASCII + \uFFFF Unicode Escapes" encoding for such files and displays the actual characters rather than the escapes. By default, this option is off for all file types except the Java file type. In Java source code files, Unicode escapes are 100% equivalent to actual characters, even for ASCII characters.

## Convert|\uFFFF -> Characters

Select the \uFFFF -> Characters command in the Convert menu to replace all escape sequence in the form of `\uFFFF` into their actual characters. If you select some text before using this command, only escape sequences in the selection are replaced. Otherwise, all escape sequences in the file are replaced.

The conversion will only be successful if the all the characters can be represented in the encoding used by the file. If not, use Convert|Text Encoding to change the encoding before converting the escapes. Escapes that represent characters that are not supported by the file's encoding will not be converted. EditPad will pop up a message to alert you when this happens.

Use the \uFFFF -> Characters command when you actually want to remove the escapes from your file. Do not use this command if you have a file that is pure ASCII with `\uFFFF` escapes and you want to be able to see the actual characters while keeping Unicode escapes in the file. To do that, use Convert|Text Encoding and select the "ASCII + \uFFFF Unicode Escapes" encoding. This encoding makes EditPad Pro display the actual characters, but keeps the escapes in the file. If you type or paste non-ASCII characters while using the "ASCII + \uFFFF Unicode Escapes" encoding, they will be automatically stored as Unicode escapes in the file.

## Convert|Characters -> &#65535;

Select the Characters -> &#65535; command in the Convert menu to replace all non-ASCII characters in the file with a decimal character reference in the form of `&#65535;`. If you select some text before using this command, only non-ASCII characters in the selection are replaced. Otherwise, all non-ASCII characters in the file are replaced.

Using this command does not change the encoding EditPad Pro uses for this file. Saving the file after replacing all non-ASCII characters this way only results in an ASCII file if the encoding you're using for the file is compatible with ASCII. For example, the Windows code pages will result in an ASCII file, but UTF-16 will not.

In Convert|Text Encoding you can select "ASCII + &#65535; NCR" as the file's encoding. When using this encoding, EditPad Pro displays the actual characters for all `&#65535;` character references in the file. When you type non-ASCII characters into the file, EditPad Pro will display the character you typed, but store its `&#65535;` NCR in the file. If you type "après-ski" using this encoding, EditPad Pro will display "après-ski", but store "après-ski" in the file. If you open that file in Notepad (which cannot interpret numeric character references), you'll see "après-ski".

Use the Convert|Characters -> &#65535; along with an encoding such as UTF-8 or Windows 1252 when you want to use numeric character references in only part of the file. EditPad will display the numeric character references while you edit the file. You can see where numeric character references are used and where actual characters are used.

Use the "ASCII + &#65535; NCR" encoding when working with files where numeric character references are the only way of representing non-ASCII characters. EditPad will display the actual characters while you edit the file. You won't see the numeric character references. EditPad makes sure all newly typed or pasted characters are stored as numeric character references in the file.

If you have a file that contains non-ASCII characters that are not written as numeric character references then you can convert that file you being pure ASCII + &#65535; NCR in three steps. First, open the file in EditPad and make sure all characters are displayed correctly. If not, use Convert|Text Encoding to select the proper encoding with the "interpret" option. Second, use Convert|Characters -> &#65535; to convert all non-ASCII characters to numeric character references. The actual escapes will appear. Finally, use Convert|Text Encoding and select to convert to ASCII + &#65535; NCR. The numeric character references

now again appear as actual characters. If you save the file the, it will be saved as pure ASCII with numeric character references.

In the Encoding section in the file type configuration, there is an option to make EditPad Pro detect whether a file consists of pure ASCII with numeric character references. If that option is on, EditPad Pro automatically selects the "ASCII + &#65535; NCR" encoding for such files and displays the actual characters rather than the numeric character references.

## Convert|Characters -> &#xFFFF;

This command does the same as Convert|Characters -> &#65535; except that it replaces characters with hexadecimal references instead of decimal references. So “après-ski” becomes "après-ski".

## Convert|&#65535; and &#xFFFF; -> Characters

Select the &#65535; and &#xFFFF; -> Characters command in the Convert menu to replace all decimal and hexadecimal character references in the form of `&#65535;` and `&#xFFFF;` into their actual characters. If you select some text before using this command, only numeric character references in the selection are replaced. Otherwise, all numeric character references in the file are replaced.

This command does not convert numeric character references that represent the characters <, >, ', ", and & that have special meanings in XML. To convert these, use Convert|XML Entities -> Reserved Characters.

The conversion will only be successful if the all the characters can be represented in the encoding used by the file. If not, use Convert|Text Encoding to change the encoding before converting the numeric character references. numeric character references that represent characters that are not supported by the file’s encoding will not be converted. EditPad will pop up a message to alert you when this happens.

Use the &#65535; and &#xFFFF; -> Characters command when you actually want to remove the numeric character references from your file. Do not use this command if you have a file that is pure ASCII with numeric character references and you want to be able to see the actual characters while keeping numeric character references in the file. To do that, use Convert|Text Encoding and select the "ASCII + NCR" encoding. This encoding makes EditPad Pro display the actual characters, but keeps the numeric character references in the file. If you type or paste non-ASCII characters while using the "ASCII + NCR" encoding, they will be automatically stored as numeric character references in the file.

## Convert|Characters -> &htmlchar;

Select the Characters -> &htmlchar; command in the Convert menu to replace all characters that have HTML entities with those HTML entities. It replaces “après-ski” with "après-ski". If you select some text before using this command, only characters in the selection are replaced. Otherwise, all characters in the file that have HTML entities are replaced.

Characters that do not have HTML entities are not changed. If you want to convert your file into pure ASCII using HTML entities and numeric character references for non-ASCII characters, first use the Characters -> &htmlchar; command to replace those characters that have named HTML entities. Then use the Characters -

> &#65535; or Characters -> &#xFFFF; command to replace the remaining non-ASCII characters that do not have named HTML entities.

This command does not replace reserved characters such as <. Thus you can safely use this command on an entire HTML file without destroying its HTML tags. If you do want to replace reserved characters such as < with entities such as <, use the Convert|Reserved Characters -> XML Entities menu item.

## Convert|&htmlchar; -> Characters

Select the &htmlchar; -> Characters command in the Convert menu to replace all named HTML entities into their actual characters. If you select some text before using this command, only HTML entities in the selection are replaced. Otherwise, all HTML entities in the file are replaced.

The conversion will only be successful if the all the characters can be represented in the encoding used by the file. If not, use Convert|Text Encoding to change the encoding before converting the HTML entities. HTML entities that represent characters that are not supported by the file's encoding will not be converted. EditPad will pop up a message to alert you when this happens.

This command does not replace numeric character references. Only named HTML entities are replaced. To replace both named HTML entities and numeric character references, use both the &htmlchar; -> Characters command and the Convert|&#65535; and &#xFFFF; -> Characters command.

This command also does not replace HTML entities that represent reserved characters such as <. To replace those, use Convert|XML Entities -> Reserved Characters.

Use the &htmlchar; -> Characters command when you actually want to remove the HTML entities from your file. Do not use this command if you have a file that is pure ASCII with HTML entities and you want to be able to see the actual characters while keeping HTML entities in the file. To do that, use Convert|Text Encoding and select the "ASCII + &htmlchar;" encoding. This encoding makes EditPad Pro display the actual characters, but keeps the HTML entities in the file. If you type or paste non-ASCII characters while using the "ASCII + NCR" encoding, they will be automatically stored as HTML entities in the file.

## Convert|Reserved Characters -> XML Entities

Select the Reserved Characters -> XML Entities command in the Convert menu to replace the characters <, >, ', ", and & that have special meanings in XML with their respective XML entities <, >, ', ", and &. If you select some text before using this command, only characters in the selection are replaced. Otherwise, all reserved characters in the file are replaced.

## Convert|XML Entities -> Reserved Characters

Select the XML Entities -> Reserved Characters command in the Convert menu to replace XML entities <, >, ', ", and & with the characters <, >, ', ", and & that they represent. Numeric character references that represent these 5 characters are also converted. If you select some text before using this command, only characters in the selection are replaced. Otherwise, all reserved characters in the file are replaced.

## Convert | ROT-13

Applies ROT-13 character rotation to the current selection, making it a little difficult to read. Applying ROT-13 again restores the original text.

Before the Internet went commercial, people had the good habit to use ROT-13 to conceal possible offensive postings in newsgroups and then add a warning message in plain text. This made sure that nobody would read the text by accident and be offended. If one was offended after applying ROT-13, he or she would only have himself or herself to blame for ignoring the warning.

# 13. Options Menu

## Options|File Type

When you open a file, EditPad looks at the file's extension to see which type of file it is. If you open a file with an extension not associated with any file type in EditPad's file type configuration, you can change the file type through Options|File Type.

Changing the file type will cause EditPad to change the file's settings for auto indent, line numbering, word wrap, etc. to those specified in the file type configuration for that file type. It will not change the file's extension. Use File|Rename and Move to change the file's name and extension.

## Options|Auto Indent

Turn on Options|Auto Indent if you want the next line to automatically start at the same column position as the previous line whenever you press Enter on the keyboard while in the editor. EditPad will accomplish this by counting the number of spaces and tabs at the beginning of the previous line and inserting them at the beginning of the new line you created by pressing Enter. This is most useful for writing source code.

When Auto Indent is on, using the Backspace key when there is only whitespace to the left of the cursor either deletes a single space, or deletes multiple spaces to go back to the previous level of indentation. The "backspace unindents" option in the Cursor Preferences determines this.

Note that "auto indent" is a file type specific setting in EditPad. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Line Numbers

Turn on Options|Lines Numbers to have EditPad display a number before each line in the left margin.

In File Type Editor Options, you can indicate if you want visible lines or physical lines to be numbered. This setting cannot be changed in the Options menu.

Note that "lines numbers" is a file type specific setting in EditPad. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Column Numbers

Turn on Options|Column Numbers to have EditPad display a horizontal ruler above the file indicating column numbers. Column numbers are only displayed when using a fixed-width font or a monospaced text layout.

Note that "column numbers" is a file type specific setting in EditPad. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Visualize Spaces

Turn on Options|Visualize Spaces if you want spaces and tabs to be visualized. Spaces will be indicated by a vertically centered dot. Tabs are indicated by a » character. This option is useful when working with files where extraneous whitespace can lead to problems, or where the difference between tabs and spaces matters.

You can set the color of visualized spaces in the Colors Preferences. If you set a background color other than "default" for whitespace in the preferences, then spaces will always be drawn with that background color, whether you've turned on the option to visualize them or not. Spaces will then appear as colored rectangles, with or without the symbol.

Note that "visualize spaces" is a file type specific setting in EditPad Pro. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Visualize Line Breaks

Turn on Options|Visualize Line Breaks if you always want to see a ¶ character at the end of each paragraph, indicating the invisible line break character(s). You can set the color of the paragraph symbols in File Types|Colors. Instead of showing the generic ¶ character for all line breaks, EditPad Pro can show specific symbols that indicate the line break style. You can configure this in the Editor Preferences.

If you turn off Options|Visualize Line Breaks, line break symbols are only displayed when line break characters are selected. If you never want to see any line break symbols, turn off "always visualize line breaks in selected text" in the Editor Preferences.

Note that "visualize line breaks" is a file type specific setting in EditPad Pro. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for

the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Word Wrap

Select Options|Word Wrap from the menu to toggle word wrapping on or off. By default, word wrap takes place at the right edge of the EditPad window. If you want the text to wrap at a certain number of characters, use the drop-down menu of the word wrap button on the toolbar. You can select one of a few commonly used line lengths, or define your own.

Note that "word wrap" is a file type specific setting in EditPad. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

Word wrapping is only used in text mode. In hexadecimal mode, use Options|Record Size.

## Options|Indent Wrapped Lines

When word wrap is on, you can select Options|Word Wrap from the menu to toggle wrapping line indentation on or off. When on, lines created by the word wrapping will be indented by the same amount as the line they were broken off from. If you turn it off, wrapped lines will start at the first column regardless of the indentation of the original line. Turning on this option is useful when turning on word wrap for files using nested block structures, such as XML files and various programming languages.

Note that "indent wrapped lines" is a file type specific setting in EditPad. If you change the setting through the Options menu or the toolbar, it is applied to the active file only. If you create a new file or open an existing one, the setting made in File Type Editor Options for the type of file you are creating or opening, will be used. Also, when you use File|Save As, and you save the file with a new extension, then the setting for the current file will change into what you specified in File Types Preferences for the file type that corresponds with the file's new extension.

## Options|Record Size

What Options|Word Wrap is to text editing, Options|Record Size is to hexadecimal editing. Use this menu item to change the number of bytes that EditPad Pro displays on each row in hexadecimal mode. If you enter zero, you get EditPad Pro's default behavior of showing the smallest multiple of 8 bytes that fits within the width of EditPad's window. If you enter a positive number, that's the number of bytes EditPad Pro displays on a row. You can enter any number. It doesn't have to be divisible by 8 or by 2.

The Record Size command is particularly useful for editing binary files in which each record takes up a specific number of bytes. Set the record size to that number of bytes to have one record on each line.

## Options|Font

If you pick Options|Font from the menu, a font selection window will appear. Select the font you want to use and click OK. You can also select a recently used font from the Options|Font submenu.

The font setting applies to the entire file. It only changes the way it is displayed. EditPad is a plain text editor. It does not support any text formatting.

Changing the font via Options|Font only affects the active file. In EditPad, the choice of font is part of the text layout configuration. That is the combination of settings that determines how EditPad displays text. The default text layout configuration can be set for each file type in File Types|Editor Options.

If you want to edit text files in a language that uses characters that are not used in English, you will need to select the correct "script" from the drop-down list near the bottom of the font selection window that appears when you select Options|Font from the menu. Not all fonts support all scripts. Particularly scripts like Arabic, Chinese, Japanese or Korean are often supported by few fonts. If you can't find any font with the script you want, you will need to install them from your original Windows CD. You can do so by clicking on the Windows Start button, choosing Settings, and opening the Control Panel. Among the Regional Settings, there are options for enabling support for complex scripts and ideographic scripts.

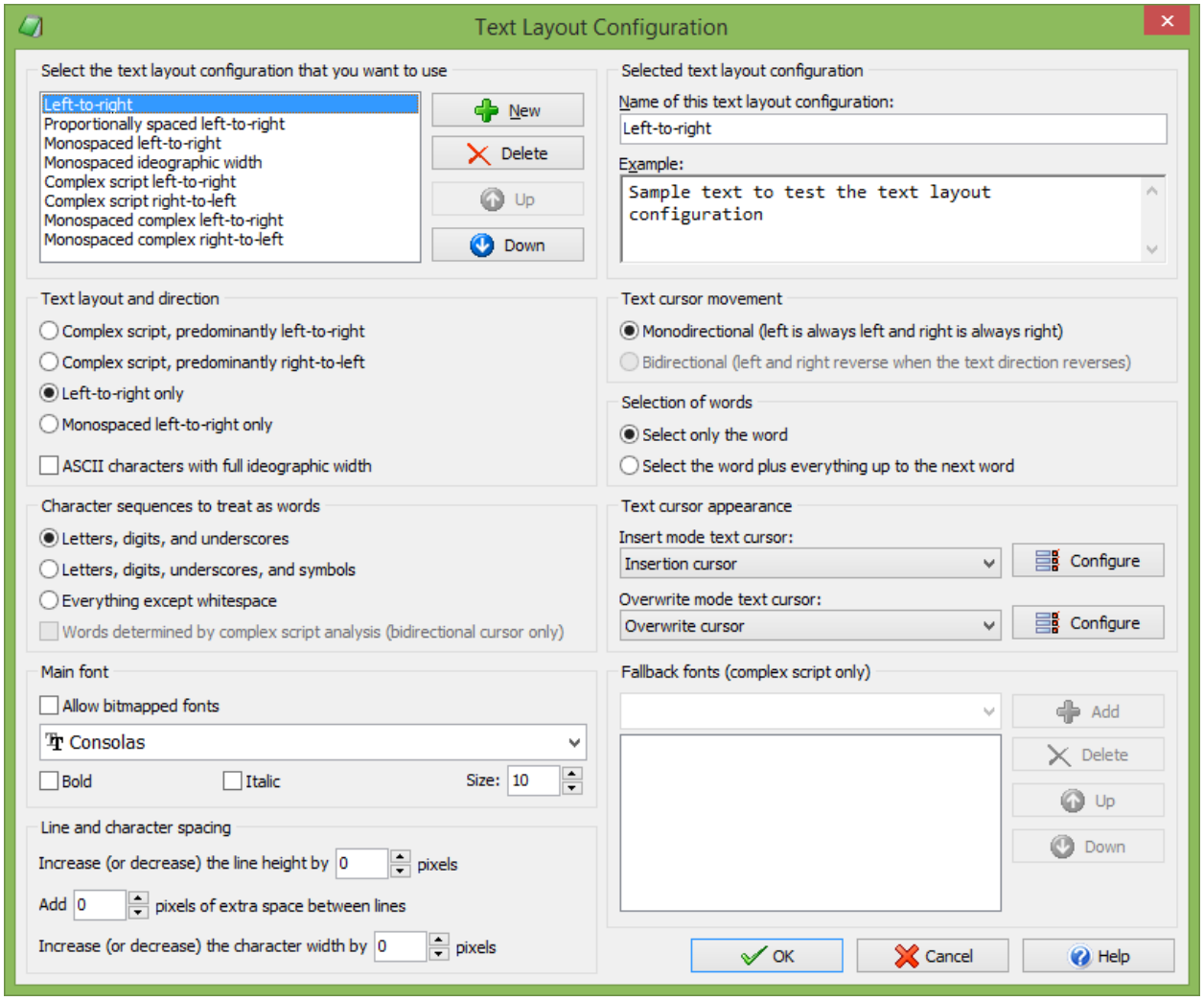
Some fonts have "Unicode" in their name. Such fonts include "Arial Unicode" and "Lucida Sans Unicode". These fonts contain all characters in all languages that Windows supports. If you use one of these fonts, you never have to worry about using the right font for the right language.

Fonts with Chinese, Japanese and Korean characters are available in two variants: the regular variant (e.g. MS Mincho), and a rotated variant for vertical printing (e.g @MS Mincho). The rotated variant always has the same name as the regular variant, but with an @ symbol in front of it. In Options|Font, you should select the regular variant, so characters will appear upright on the screen. In the print preview, you can select the rotated variant if you want. If you print with the rotated variant, the text will appear printed vertically if you rotate the printed sheet of paper 90 degrees clockwise. If you print with the regular variant, the printed text will flow from left to right, like you edit it in EditPad.

## Options|Text Layout

In EditPad, a "text layout" is a combination of settings that control how text is displayed and how the text cursor navigates through that text. The settings include the font, text direction, text cursor behavior, which characters are word characters, and how the text should be spaced.

The default text layout is configured for each file type in File Types|Editor Options. Via the Text Layout submenu in the Options menu you can select a different text layout for the active file. You can use the Configure Text Layout item in the Text Layout submenu to configure a new text layout or to edit an existing one. If you edit an existing text layout, the changes will be applied to any file and to any file type that uses it.




## Select The Text Layout Configuration That You Want to Use

The Text Layout Configuration screen shows the details of the text layout configuration that you select in the list in the top left corner. Any changes you make on the screen are automatically applied to the selected layout and persist as you choose different layouts in the list. The changes become permanent when you click OK. The layout that is selected in the list when you click OK becomes the new default layout.

Click the New and Delete buttons to add or remove layouts. You must have at least one text layout configuration. If you have more than one, you can use the Up and Down buttons to change their order. The order does not affect anything other than the order in which the text layouts configurations appear in selection lists.

EditPad comes with a number of preconfigured text layouts. If you find the options on this screen bewildering, simply choose the preconfigured layout that matches your needs, and ignore all the other settings. You can fully edit and delete all the preconfigured text layouts if you don't like them.

- Left-to-right: Normal settings with best performance for editing text in European languages and ideographic languages (Chinese, Japanese, Korean). The default font is monospaced. The layout does respect individual character width if the font is not purely monospaced or if you select another font.
- Proportionally spaced left-to-right: Like left-to-right, but the default font is proportionally spaced.
- Monospaced left-to-right: Like left-to-right, but the text is forced to be monospaced. Columns are guaranteed to line up perfectly even if the font is not purely monospaced. This is the best choice for working with source code and text files with tabular data.
- Monospaced ideographic width: Like monospaced left-to-right, but ASCII characters are given the same width as ideographs. This is the best choice if you want columns of mixed ASCII and ideographic text to line up perfectly.
- Complex script left-to-right: Supports text in any language, including complex scripts (e.g. Indic scripts) and right-to-left scripts (Hebrew, Arabic). Choose this for editing text that is written from left-to-right, perhaps mixed with an occasional word or phrase written from right-to-left.
- Complex script right-to-left: For writing text in scripts such as Hebrew or Arabic that are written from right-to-left, perhaps mixed with an occasional word or phrase written from left-to-right.
- Monospaced complex left-to-right: Like "complex script left-to-right", but using monospaced fonts for as many scripts as possible. Text is not forced to be monospaced, so columns may not line up perfectly.
- Monospaced complex right-to-left: Like "complex script right-to-left", but using monospaced fonts for as many scripts as possible. Text is not forced to be monospaced, so columns may not line up perfectly.

## Selected Text Layout Configuration

The section in the upper right corner provides a box to type in the name of the text layout configuration. This name is only used to help you identify it in selection lists when you have prepared more than one text layout configuration.

In the Example box you can type in some text to see how the selected text layout configuration causes the editor to behave.

## Text Layout and Direction

- Complex script, predominantly left-to-right: Text is written from left to right and can be mixed with text written from right to left. Choose this for complex scripts such as the Indic scripts, or for text in any language that mixes in the occasional word or phrase in a right-to-left or complex script.
- Complex script, predominantly right-to-left: Text is written from right to left and can be mixed with text written from left to right. Choose this for writing text in scripts written from right to left such as Hebrew or Arabic.
- Left-to-right only: Text is always written from left to right. Complex scripts and right-to-left scripts are not supported. Choose this for best performance for editing text in European languages and ideographic languages (Chinese, Japanese, Korean) that is written from left to right without exception.
- Monospaced left-to-right only: Text is always written from left to right and is forced to be monospaced. Complex scripts and right-to-left scripts are not supported. Each character is given the same horizontal width even if the font specifies different widths for different characters. This guarantees columns to be lined up perfectly. To keep the text readable, you should choose a monospaced font.

- ASCII characters with full ideographic width: You can choose this option in combination with any of the four preceding options. In most fonts, ASCII characters (English letters, digits, and punctuation) are about half the width of ideographs. This option substitutes full-width characters for the ASCII characters so they are the same width as ideographs. If you turn this on in combination with "monospaced left-to-right only" then columns that mix English letters and digits with ideographs will line up perfectly.

## Text Cursor Movement

- Monodirectional: The left arrow key on the keyboard always moves the cursor to the left on the screen and the right arrow key always moves the cursor to the right on the screen, regardless of the predominant or actual text direction.
- Bidirectional: This option is only available if you have chosen one of the complex script options in the "text layout and direction" list. The direction that the left and right arrow keys move the cursor into depends on the predominant text direction selected in the "text layout and direction" list and on the actual text direction of the word that the cursor is pointing to when you press the left or right arrow key on the keyboard.
    - Predominantly left-to-right: The left key moves to the preceding character in logical order, and the right key moves to the following character in logical order.
        - Actual left-to-right: The left key moves left, and the right key moves right.
        - Actual right-to-left: The actual direction is reversed from the predominant direction. The left key moves right, and the right key moves left.
    - Predominantly right-to-left: The left key moves to the following character in logical order, and the right key moves to the preceding character in logical order.
        - Actual left-to-right: The actual direction is reversed from the predominant direction. The left key moves right, and the right key moves left.
        - Actual right-to-left: The left key moves left, and the right key moves right.

## Selection of Words

- Select only the word: Pressing Ctrl+Shift+Right moves the cursor to the end of the word that the cursor is on. The selection stops at the end of the word. This is the default behavior for all Just Great Software applications. It makes it easy to select a specific word or string of words without any extraneous spaces or characters. To include the space after the last word, press Ctrl+Shift+Right once more, and then Ctrl+Shift+Left.
- Select the word plus everything to the next word: Pressing Ctrl+Shift+Right moves the cursor to the start of the word after the one that the cursor is on. The selection includes the word that the cursor was on and the non-word characters between that word and the next word that the cursor is moved to. This is how text editors usually behave on the Windows platform.

## Character Sequences to Treat as words

- Letters, digits, and underscores: Characters that are considered to be letters, digits, or underscores by the Unicode standard are selected when you double-click them. Ctrl+Left and Ctrl+Right move the cursor to the start of the preceding or following sequence of letters, digits, or underscores. If symbols

or punctuation appear adjacent to the start of a word, the cursor is positioned between the symbol and the first letter of the word. Ideographs are considered to be letters.

- Letters, digits, and symbols: As above, but symbols other than punctuation are included in the selection when double-clicking. Ctrl+Left and Ctrl+Right never put the cursor between a symbol and another word character.
- Everything except whitespace: All characters except whitespace are selected when you double-click them. Ctrl+Left and Ctrl+Right move the cursor to the preceding or following position that has a whitespace character to the left of the cursor and a non-whitespace character to the right of the cursor.
- Words determined by complex script analysis: If you selected the "bidirectional" text cursor movement option, you can turn on this option to allow Ctrl+Left and Ctrl+Right to place the cursor between two letters for languages such as Thai that don't write spaces between words.

## Text Cursor Appearance

Select a predefined cursor or click the Configure button to show the text cursor configuration screen. There you can configure the looks of the blinking text cursor (and even make it stop blinking).

A text layout uses two cursors. One cursor is used for insert mode, where typing in text pushes ahead the text after the cursor. The other cursor is used for overwrite mode, where tying in text replaces the characters after the cursor. Pressing the Insert key on the keyboard toggles between insert and overwrite mode.

## Main Font

Select the font that you want to use from the drop-down list. Turn on "allow bitmapped fonts" to include bitmapped fonts in the list. Otherwise, only TrueType and OpenType fonts are included. Using a TrueType or OpenType font is recommended. Bitmapped fonts may not be displayed perfectly (e.g. italics may be clipped) and only support a few specific sizes.

If you access the text layout configuration screen from the print preview, then turning on "allow bitmapped fonts" will include printer fonts rather than screen fonts in the list, in addition to the TrueType and OpenType fonts that work everywhere. A "printer font" is a font built into your printer's hardware. If you select a printer font, set "text layout and direction" to "left to right only" for best results.

## Fallback Fonts

Not all fonts are capable of displaying text in all scripts or languages. If you have selected one of the complex script options in the "text layout and direction" list, you can specify one or more "fallback" fonts. If the main font does not support a particular script, EditPad will try to use one of the fallback fonts. It starts with the topmost font at the list and continues to attempt fonts lower in the list until it finds a font that supports the script you are typing with. If none of the fonts supports the script, then the text will appear as squares.

To figure out which scripts a particular font supports, first type or paste some text using those scripts into the Example box. Make sure one of the complex script options is selected. Then remove all fallback fonts. Now you can change the main font and see which characters that font can display. When you've come up with a

list of fonts that, if used together, can display all of your characters, select your preferred font as the main font. Then add all the others as fallback fonts.

## Line and Character Spacing

By default all the spacing options are set to zero. This tells EditPad to use the default spacing for the font you have selected.

If you find that lines are spaced apart too widely, specify a negative value for "increase (or decrease) the line height". Set to “add 0 pixels of extra space between lines”.

If you find that lines are spaced too closely together, specify a positive value for "increase (or decrease) the line height" and/or "add ... pixels of extra space between lines". The difference between the two is that when you select a line of text, increasing the line height increases the height of the selection highlighting, while adding extra space between lines does not. If you select multiple lines of text, extra space between lines shows up as gaps between the selected lines. Adding extra space between lines may make it easier to distinguish between lines.

The "increase (or decrease) the character width by ... pixels“ setting is only used when you select ”monospaced left-to-right“ only in the ”text layout and direction" list. You can specify a positive value to increase the character or column width, or a negative value to decrease it. This can be useful if your chosen font is not perfectly monospaced and because of that characters appear spaced too widely or too closely.

## Text Cursor Configuration

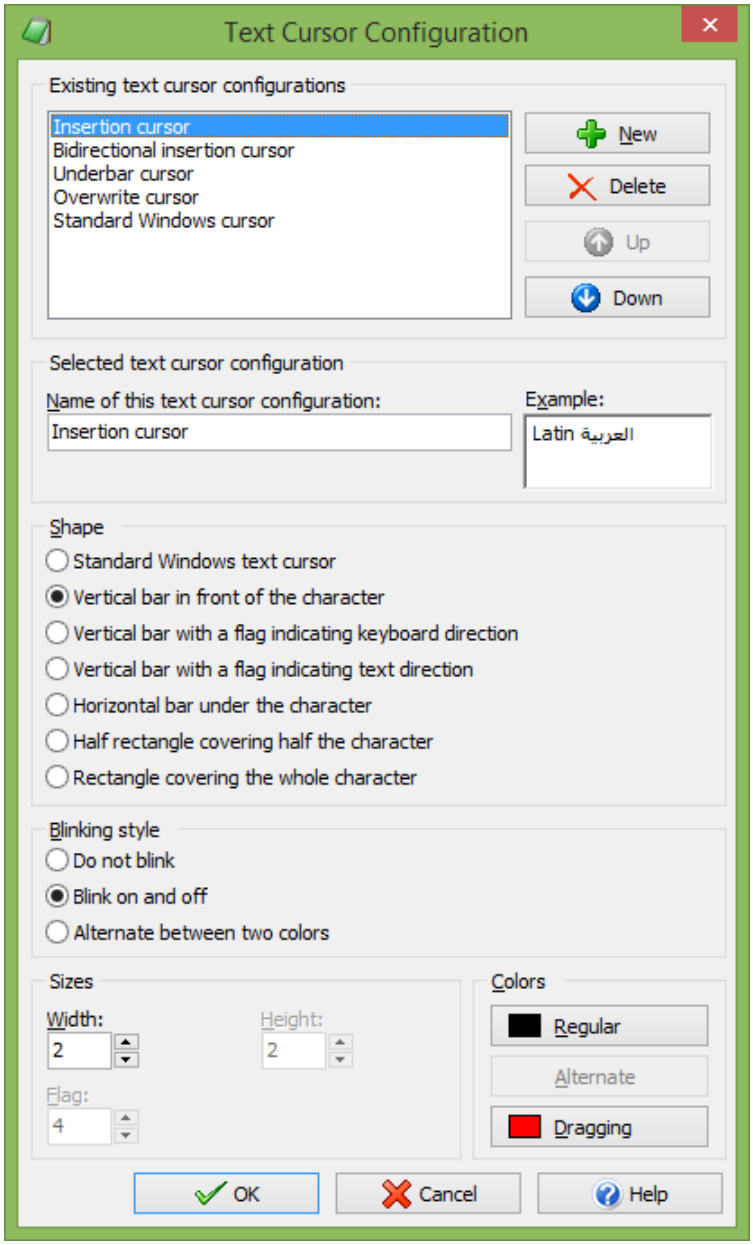
You can access the text cursor configuration screen from the text layout configuration screen by clicking one of the Configure buttons in the “text cursor appearance” section.

## Existing Text Cursor Configurations

The Text Cursor Configuration screen shows the details of the text cursor configuration that you select in the list at the top. Any changes you make on the screen are automatically applied to the selected cursor and persist as you choose different cursors in the list. The changes become permanent when you click OK. The cursor that is selected in the list when you click OK becomes the new default cursor.

Click the New and Delete buttons to add or remove cursors. You must have at least one text cursor configuration. If you have more than one, you can use the Up and Down buttons to change their order. The order does not affect anything other than the order in which the text cursor configurations appear in selection lists.

EditPad comes with a number of preconfigured text cursors. You can fully edit or delete all the preconfigured text cursors if you don’t like them.




of EditPad's other cursor shapes.

- Insertion cursor: Blinking vertical bar similar to the standard Windows cursor, except that it is thicker and fully black, even on a gray background.
- Bidirectional insertion cursor: Like the insertion cursor, but with a little flag that indicates whether the keyboard layout is left-to-right (e.g. you're tying in English) or right-to-left (e.g. you're typing in Hebrew). The flag is larger than what you get with the standard Windows cursor and is shown even if you don't have any right-to-left layouts installed.
- Underbar cursor: Blinking horizontal bar that lies under the character. This mimics the text cursor that was common in DOS applications.
- Overwrite cursor: Blinking rectangle that covers the bottom half of the character. In EditPad this is the default cursor for overwrite mode. In this mode, which is toggled with the Insert key on the keyboard, typing text overwrites the following characters instead of pushing them ahead.
- Standard Windows cursor: The standard Windows cursor is a very thin blinking vertical bar that is XOR-ed on the screen, making it very hard to see on anything except a pure black or pure white background. If you have a right-to-left keyboard layout installed, the cursor gets a tiny flag indicating keyboard direction. You should only use this cursor if you rely on accessibility software such as a screen reader or magnification tool that fails to track any

## Selected Text Cursor Configuration

Type in the name of the text cursor configuration. This name is only used to help you identify it in selection lists when you have prepared more than one text cursor configuration.

In the Example box you can type in some text to see what the cursor looks like. The box has a word in Latin and Arabic so you can see the difference in cursor appearance, if any, based on the text direction of the word that the cursor is on.

## Shape

- Standard Windows Text cursor: The standard Windows cursor is a very thin blinking vertical bar that is XOR-ed on the screen, making it very hard to see on anything except a pure black or pure white background. If you have a right-to-left keyboard layout installed, the cursor gets a tiny flag indicating keyboard direction. You should only use this cursor if you rely on accessibility software such as a screen reader or magnification tool that fails to track any of EditPad's other cursor shapes. The standard Windows cursor provides no configuration options.
- Vertical bar in front of the character: On the Windows platform, the normal cursor shape is a vertical bar that is positioned in front of the character that it points to. That is to the left of the character for left-to-right text, and to the right of the character for right-to-left text.
- Vertical bar with a flag indicating keyboard direction: A vertical bar positioned in front of the character that it points to, with a little flag (triangle) at the top that indicates the direction of the active keyboard layout. When the cursor points to a character in left-to-right text, it is placed to the left of that character. When the cursor point to a character in right-to-left text, it is placed to the right of that character. The direction of the cursor's flag is independent of the text under the cursor. The cursor's flag points to the right when the active keyboard layout is for a left-to-right language. The cursor's flag points to the left when the active keyboard layout is for a right-to-left language.
- Vertical bar with a flag indicating text direction: A vertical bar positioned in front of the character that it points to, with a little flag (triangle) at the top that points to that character. When the cursor points to a character in left-to-right text, it is placed to the left of that character with its flag pointing to the right towards that character. When the cursor point to a character in right-to-left text, it is placed to the right of that character with its flag pointing to the left towards that character.
- Horizontal bar under the character: In DOS applications, the cursor was a horizontal line under the character that the cursor points to.
- Half rectangle covering half the character: The cursor covers the bottom half of the character that it points to. This is a traditional cursor shape to indicate typing will overwrite the character rather than push it ahead.
- Rectangle covering the whole character: The cursor makes the character invisible. This can also be used to indicate overwrite mode.

## Blinking Style

- Do not blink: The cursor is permanently visible in a single color. Choose this option if the blinking distracts you or if it confuses accessibility software such as screen readers or magnification tools.
- Blink on and off: The usual blinking style for text cursors on the Windows platform. The cursor is permanently visible while you type (quickly). When you stop typing for about half a second, the cursor blinks by becoming temporarily invisible. Blinking makes it easier to locate the cursor with your eyes in a large block of text.
- Alternate between two colors: Makes the cursor blink when you stop typing like "on and off". But instead of making the cursor invisible, it is displayed with an alternate color. This option gives the cursor maximum visibility: the blinking animation attracts the eye while keeping the cursor permanently visible.

## Sizes

- Width: Width in pixels for the vertical bar shape.

- Height: Height in pixels for the horizontal bar shape.
- Flag: Length in pixels of the edges of the flag that indicates text direction.

## Colors

- Regular: Used for all shapes and blinking styles except the standard Windows cursor.
- Alternate: Alternate color used by the "alternate between two colors" blinking style.
- Dragging: Color of a second "ghost" cursor that appears while dragging and dropping text with the mouse. It indicates the position the text is moved or copied to when you release the mouse button.

## Options | Right-to-Left

Select Right-to-Left in the Options menu to quickly toggle the active file between left-to-right and right-to-left editing. What this command really does is switch between the "default text layout" and the "layout for opposite text direction" that you selected for the active file's file type in File Types | Editor Options.

If you have at least one right-to-left keyboard layout installed in Windows, then you can also do this with the Ctrl and Shift keys on the keyboard. To switch to left-to-right, hold down either Ctrl key while pressing and releasing the left hand Shift key. To switch to right-to-left, hold down either Ctrl key while pressing the right hand Shift key. These key combinations too switch between the "default text layout" and the "layout for opposite text direction".

## Options | Stay On Top

If you turn on Options | Stay On Top, EditPad's window will remain visible even when you switch to another application.

## Options | Export Preferences

Use Options | Export Preferences to save your preferences into an .ini file. This allows you to transport them to another computer.

## Options | Import Preferences

Imports preferences saved with Options | Export Preferences.

# 14. Options|Configure File Types

One of the most powerful aspects about the way EditPad Lite and Pro work with preferences is that many settings can be made for each specific file type that you regularly edit or view with EditPad. For example: you may want to use word wrapping at the window border for plain text files, but no word wrapping for source code. Once you make the correct settings for each file type, you won't have to change them each time you open a file.

When you open or save a file, EditPad compares its name against the file mask for each known file type and applies the settings made for that file type. If the file name doesn't match any file type, the settings for "unspecified file type" are used.

EditPad also uses the list of file types to build the "Files of type" drop-down list at the bottom of the file selection windows for opening and saving files. The filters in the Explorer Panel, the FTP Panel, the Open Folder command, and the Find on Disk command also allow you to select files based on their file types.

To configure file types, select Configure File Types in the Options menu. The file type configuration screen will appear. The screen is divided into two parts. At the left hand, you'll see a list of currently defined file types. At the right hand are four tabs that hold the settings for the file type that's currently selected in the list at the left.

To change a file type's settings, simply click on it in the list and make the changes you want. Hold down the Shift or Ctrl key to select multiple file types. Any changes you make are applied to all selected file types. To create a new file type, click the New button below the list and set the options as you want them. To delete a file type, click on it and click the Delete button.

The order of the file types matters. The order you give them in the configuration screen is the order they'll have in any drop-down list showing the available file types. If a file matches the file mask of more than one file type, EditPad will use the bottommost file type in the list with a matching mask for that file. Place file types with more specific file masks below those with more general file masks. The "unspecified file type" file type is the most general one, and always sits at the top of the list. The other file types defined in EditPad's default setting have no overlap in their file masks, so their order doesn't matter. They're ordered alphabetically by default. If you edit the file masks of the predefined file types or add your own file types, you may have to reorder them if the file masks of two file types can match a single file.

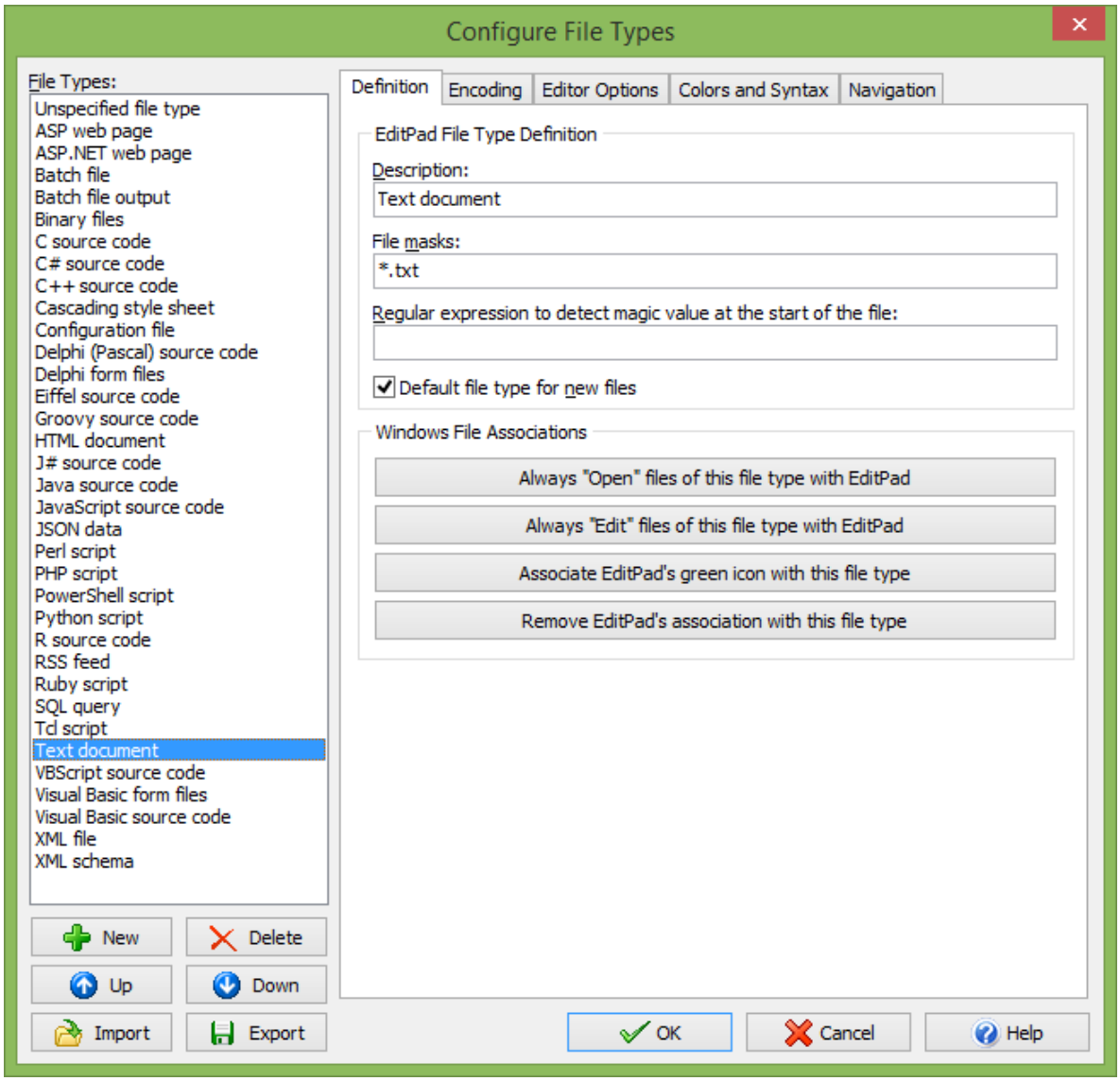
You can save individual file type configurations into a file that you can share with other people. To save a file type, click on it in the list and click the Export button below the list. If you select multiple file types, all of them are exported. To load a file type configuration file you've received from somebody, click the Import button. The loaded file type will be added to the list.

For each file type, you can specify five sets of options:

- Definition and Associations
- Encoding
- Editor Options
- Colors and Syntax
- Navigation

# File Type Definition

On the Definition tab in the file types configuration screen, you can indicate how the file type should be identified and which files it applies to.




The **description** is used whenever you need to make a choice between file types, such as the "files of type" list in open and save windows or the Options|File Type command. You cannot change the description of the first few file types. These file types have a special meaning in EditPad. You cannot delete these file types or change their position in the list, but you can adjust all the settings to your own tastes and habits as you can do for all the other file types.

The **file masks** you assign to a file type are particularly important. When you open a file, EditPad compare's the file's name against the file masks for each file type. If it finds a file type with a mask that matches the file name, that file type's settings will be used for the file being opened. If more than one file type's mask matches, the *bottom*most matching file type is used. Therefore, file types with more specific masks should be

placed below file types with more general masks. The "any file" file type has a mask of "*.*", which matches all file names. If there's no other file type with a matching mask, the "any file" file type will be used. The "any file" file type always sits at the top of the list, and is thus the last and most generic file type.

In a file mask, the asterisk (*) represents any number (including none) of any character. The question mark (?) represents one single character. On the Windows platform, the type of a file is usually determined by its name's extension. The extension consists of a dot followed by (usually) three letters. E.g. text files have a .txt extension. The file mask *.txt will match all text files with a .txt extension. You can delimit multiple file masks with semicolons. The file mask *.c;*.h matches all C source and header files (with a .c and .h extension respectively). If you specify a semicolon-delimited list of file masks, a file's name needs to match only one of them for the file type to be applied to that file.

File masks also support a simple character class notation, which matches one character from a list or a range of characters. E.g. a file mask such as www.200509[0123][0-9].log or www.200509??.log could be used to match all web logs from September 2005.

Sometimes, a file's type cannot be easily derived from its name. While file name extensions are common on the Windows platform, they're not on other platforms like UNIX. For such file types, you can specify a **magic value** regular expression. A magic value is simply some text or data at the start of a file that reveals the file's type. A regular expression is a pattern for matching text.

When EditPad has finished comparing the file's name against the file masks of all file types, and the only matching file type is "unspecified file type", EditPad will try to match the magic value regular expression of each file type at the start of the file. The regex is only attempted at the very start of the file, as if the regular expression started with the anchor «`\A`». Again, should more than one file type have a matching regular expression, the bottommost file type will be used.

Out of the box, EditPad Pro ships with several file types with magic value regular expressions. The "HTML" file type uses «`(?i)\s*<(!DOCTYPE\s+)?HTML`» to match a `<!DOCTYPE HTML` or `<HTML` tag at the start of the file, case insensitively. The "Perl script" file type has «`#![-_.\\/a-zA-Z0-9]*(?:/env )?perl`» which matches the "shebang" at the start of a Perl script. On the UNIX platform, Perl scripts usually don't have an extension, but do have the shebang. On the Windows platform, the shebang is typically missing, but Perl scripts are given a .pl extension. EditPad will recognize the file either way, first trying the file masks to look for the `.pl` extension, and then trying the regular expression to match the shebang. The "XML" file type uses «`<\?xml |\s*<[a-zA-Z0-9_:]+\s++[^<>]*?xmlns\s*=`» to match the `<?xml` declaration or a root tag with the `xmlns` attribute. XML files often have an extension that indicates the application that saved the file, rather than the fact that it's in XML format. E.g. PowerGREP uses the `.pgf`, `.pga`, `.pgr` and `.pgl` extensions for PowerGREP file selections, actions, results and libraries, rather than the generic `.xml` extension.

Check "**default for new files**" to use the file type for new files created by clicking the File menu directly or by pressing its keyboard shortcut Ctrl+N.

## File Type Associations

Using the three buttons on the Definition tab in the file types configuration screen, you can quickly create or remove file associations to the current file type. The associations will be made with or removed from each of the extensions listed in the file type's file mask.

Windows itself determines file types by file name extension only. Therefore, associations will only be made with file masks in the style of *.ext. Other file masks are ignored when making Windows file associations.

If you click the button **“Always ”Open“ this file type with EditPad”**, EditPad will associate itself with the “open” action. This means that whenever you double-click on a file of this type in Windows Explorer, it will be opened in EditPad.

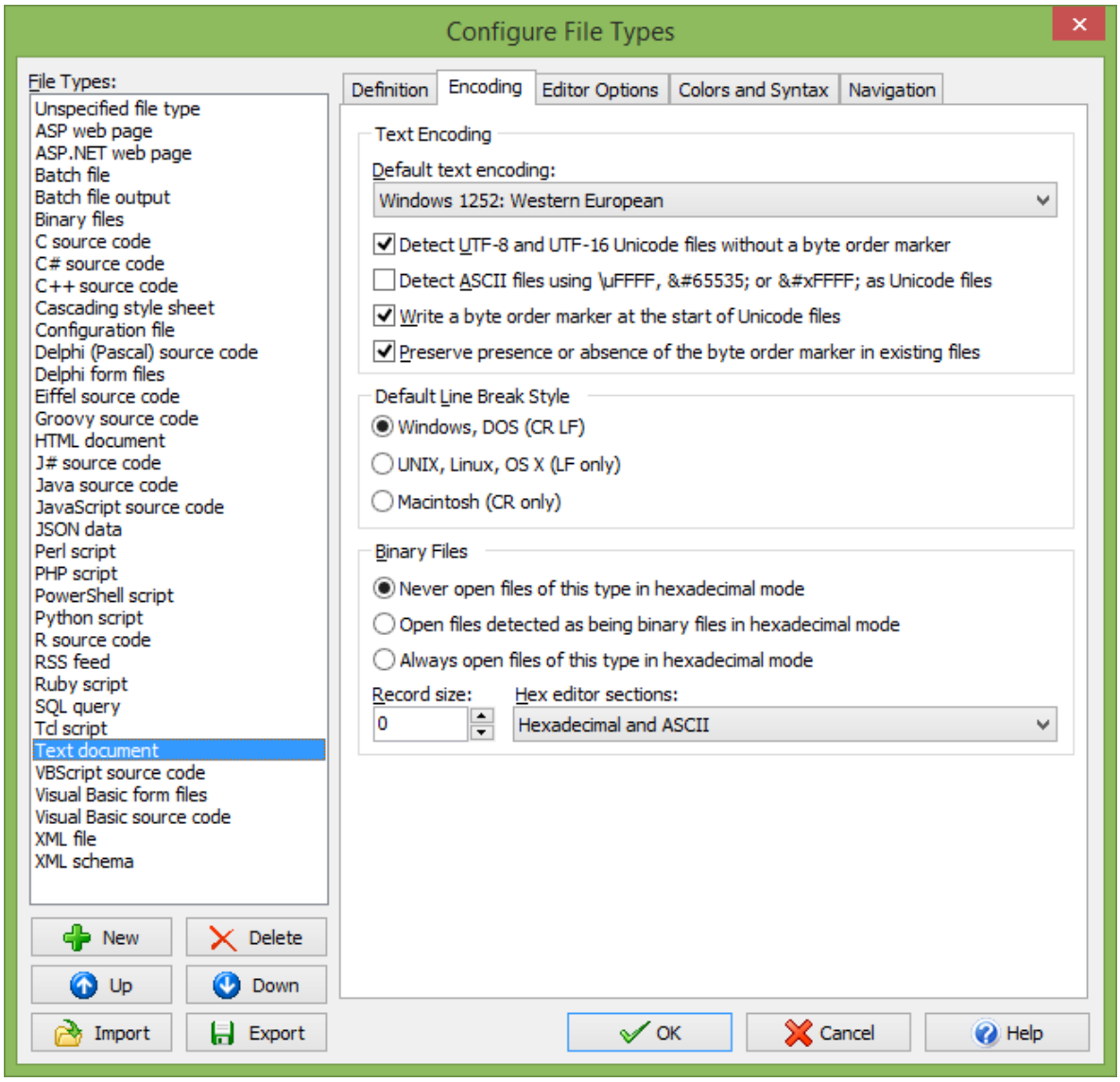
If you click the button **“Always ”Edit“ this file type with EditPad”**, EditPad will associate itself with the “edit” action. This means that you can then right-click on files of this type in Windows Explorer and select “Edit” from the context menu to open the file in EditPad. If you do this for HTML files, you will be able to view the source for any web page by clicking on the Edit button in the toolbar of Microsoft Internet Explorer.

If you click **"Associate EditPad’s green icon with this file type"**, then Windows Explorer will indicate files of this type with the green notebook icon that represents EditPad itself. You can use this button to make files that you edit exclusively with EditPad stand out in Windows Explorer or other file management applications. EditPad itself also uses the icon associated with a file type to indicate file types on tabs and file management side panels.

Click the button **"Remove EditPad’s association with this file type"** to remove the associations made by clicking on one or more of the three previous buttons. Associations made in another way will not be removed.

# File Type Encoding

On the Encoding tab in the file types configuration screen, you can indicate how EditPad should encode and decode files of a particular type.




Computers deal with numbers, not with characters. When you save a text file, each character is mapped to a number, and the numbers are stored on disk. When you open a text file, the numbers are read and mapped back to characters. When saving a file in one application, and opening the that file in another application, both applications need to use the same character mappings.

Traditional character mappings or code pages use only 8 bits per character. This means that only 256 characters can be represented in any text file. As a result, different character mappings are used for different language and scripts. Since different computer manufacturers had different ideas about how to create character mappings, there's a wide variety of legacy character mappings. EditPad supports a wide range of these.

In addition to conversion problems, the main problem with using traditional character mappings is that it is impossible to create text files written in multiple languages using multiple scripts. You can't mix Chinese, Russian and French in a text file, unless you use Unicode. Unicode is a standard that aims to encompass all traditional character mappings, and all scripts used by current and historical human languages.

If you only edit files created on your own Windows computer, or on other Windows computers using the same regional settings, there's not much to configure. Simply leave the "default text encoding" set to the default Windows code page, e.g. Windows 1252 for English and other Western European languages. If you edit files created on Windows computers with different regional settings, you may need to select a different Windows code page. You can either change the default for the file type, or use Convert|Text Encoding for a one-time change.

## Unicode

On the Windows platform, Unicode files should start with a byte order marker (BOM). The byte order marker is a special code that indicates the Unicode encoding (UTF-8, UTF-16 or UTF-32) used by the file. EditPad will always detect the byte order marker, and treat the file with the corresponding Unicode encoding.

Unfortunately, some applications cannot deal with files starting with a byte order marker. XML parsers are a notorious example. If an application that claims to support Unicode fails to read Unicode files saved by EditPad, try turning on the option not to write the byte order marker.

If you turn on the option "preserve presence or absence of the byte order marker in existing files", then EditPad will keep the BOM in files that already have it, and never add it to Unicode files previously saved without a BOM. When preserving the BOM or its absence, the "write a byte order marker at the start of Unicode files" is only used for files you newly create with EditPad, and for files that you convert from a non-Unicode encoding to Unicode. (Non-Unicode files never have a BOM, so there's no presence or absence of it to maintain.)

If you want EditPad to open Unicode files saved without a byte order marker, you'll either need to set the default encoding for the file type to the proper Unicode encoding, or turn on the option to auto-detect UTF-8 and UTF-16 files without a byte order marker.

To auto-detect UTF-8 files, EditPad Pro checks if the file contains any bytes with the high order bit set (values 0x80 through 0xFF). If it does, and all the values define valid UTF-8 sequences, the file is treated as a UTF-8 file. The chances of a normal text document written in a Windows code page being incorrectly detected as UTF-8 are practically zero. Note that files containing only English text are indistinguishable from UTF-8 when encoded in any Windows, DOS or ISO-8859 code page. This is one of the design goals of UTF-8. These files contain no bytes with the high order bit set. EditPad Pro will use the file type's default code page for such files. This makes no difference as long as you don't add text in a language that doesn't use the English alphabet.

Reading a UTF-16 file as if it was encoded with a Windows code page will cause every other character in the file to appear as a NULL character. These will show up as squares or spaces in EditPad. EditPad can detect this situation in many cases and read the file as UTF-16. However, for files containing genuine NULL characters, you may need to turn off the option to detect UTF-8 and UTF-16 files without the byte order marker.

Some file formats consist of pure ASCII with non-ASCII characters represented by Unicode escapes in the form of `\uFFFF` or by numeric character references in the form of `&#65535;` or `&#xFFFF`. EditPad Pro has

ASCII + \uFFFF and ASCII + NCR text encodings that you can use to edit such files showing the actual Unicode characters in EditPad, but saving the Unicode escapes or numeric character references in the file. Turn on "Detect &ASCII files using \uFFFF, &&#65535; or &&#xFFFF; as Unicode files" to automatically use these encodings for files that consist of pure ASCII and that contain at least one of these Unicode escapes or numeric character references. By default, this option is only on for Java source code, because in Java there is no difference between a Unicode escape and the actual Unicode character. You can also turn it on for HTML or XML if you like to write your HTML and XML files in pure ASCII with character references.

## Line Break Style

If the differences in character mappings weren't enough, different operating systems also use different characters to end lines. EditPad automatically and transparently handles all three line break styles. If you open a file, EditPad will maintain that file's line break style when you edit it. EditPad will only change the line break style if you tell it to using the Convert menu.

Unfortunately, many other applications are not as versatile as EditPad. Most applications expect a file to use the line break style of the host operating system. E.g. the Notepad applet included with Windows will display all text on one long line if a file uses UNIX line breaks. The Linux shell won't properly recognize the shebang of Perl scripts saved in Windows format (causing CGI scripts to break "mysteriously").

In such situations, you will need to set a default line break style for the affected file types. When you create a new file by selecting a file type from the drop-down menu of the new file button on the toolbar, EditPad will give that file the default line break style of the chosen file type.

Note that EditPad will never silently convert line breaks to a different style. If you set the default line break style for Perl scripts to UNIX, and then open a Perl script using Windows line breaks, EditPad will save that script with Windows line breaks unless you use Convert|To UNIX.

If you need to deal with different line break styles, you should turn on the line break style status bar indicator. EditPad will indicate the line break style being used by the current file.

## Binary Files

EditPad Pro can edit binary files in hexadecimal mode. If you know that files of a certain type don't contain (much) human-readable content, select "always open files of this type in hexadecimal mode". If you know files of a certain type to be text files, select "never open files of this type in hexadecimal mode" to prevent stray NULL characters from making EditPad think the file is binary.

If you're not sure, select "open files detected as being binary in hexadecimal mode". Then EditPad Pro will check if the file contains any NULL characters. Text files should not contain NULL characters, though improperly created text files might. Binary files frequently contain NULL characters.

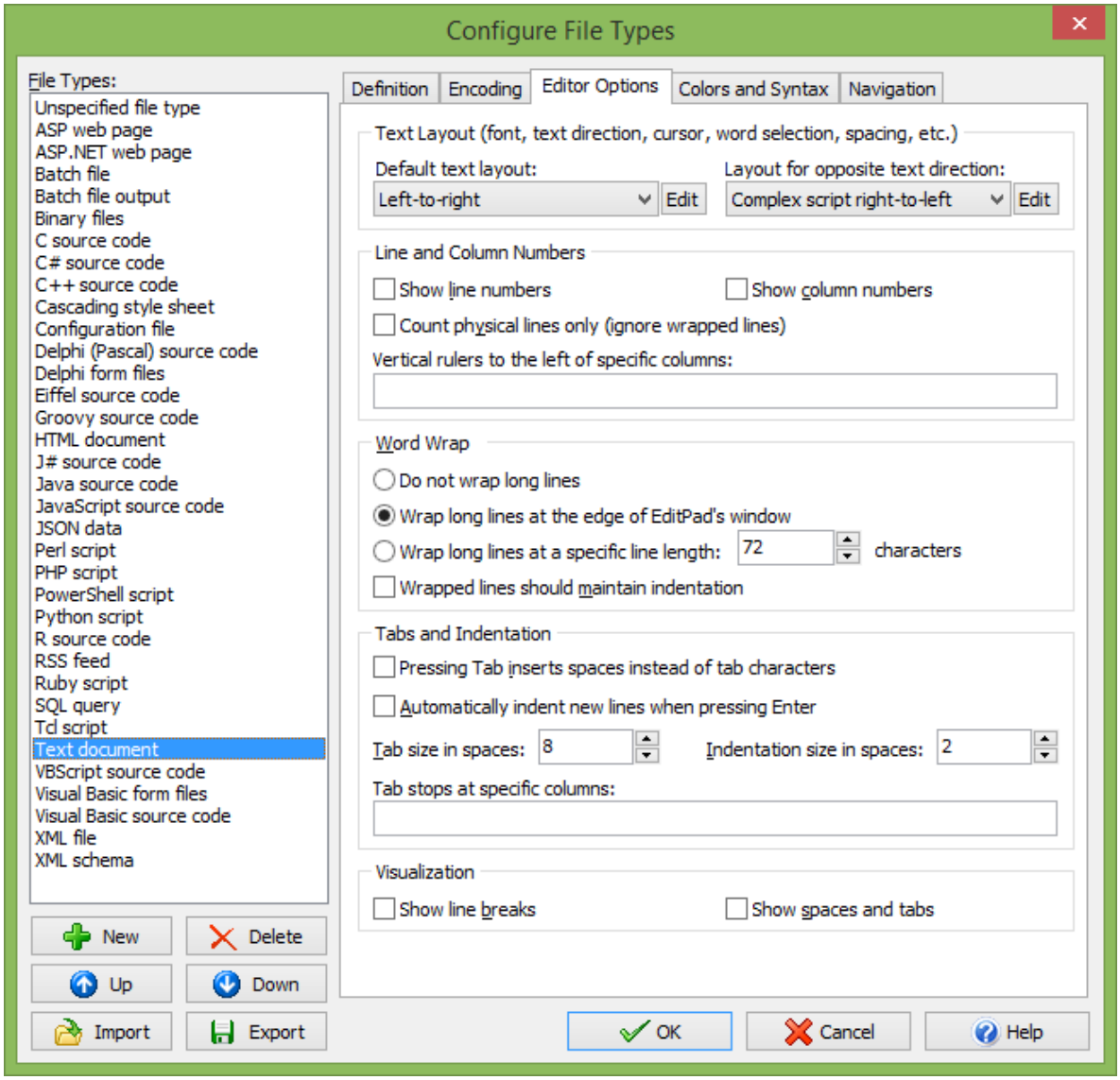
Making the wrong choice here causes no harm. You can instantly switch between text and hexadecimal mode by picking View|Hexadecimal in the menu. Unlike many other editors, EditPad Pro will preserve the exact contents of binary files even when you view them in text mode. Even files with NULL characters will be properly displayed in text mode. (Many applications truncate text files at the first NULL, since that character is often used as an end-of-data signal.)

The “record size” is the number of bytes that EditPad Pro displays on each line in hexadecimal mode. If you enter zero, you get EditPad Pro’s default behavior of showing the smallest multiple of 8 bytes that fits within the width of EditPad’s window. If you enter a positive number, that’s the number of bytes EditPad Pro displays on a line. You can enter any number. It doesn’t have to be divisible by 8 or by 2.

Set “hex editor sections” to “hexadecimal and ASCII” to get the typical hex editor view with the hexadecimal representation of the bytes in the center of the editor, and the ASCII representation of the bytes in the right hand column. Choose “hexadecimal only” or “ASCII only” to see only either representation. Select “split hexadecimal and ASCII” if you want one view to display the hexadecimal representation and the other view the ASCII representation after using View|Split Editor. If the editor is not split, there is no difference between the “split hexadecimal and ASCII” and “hexadecimal and ASCII” choices.

# File Type Editor Options

On the Editor Options tab in the file types configuration screen, you indicate the default editor options for each file type. Most of these options can be changed for individual files via the Options menu. The Options menu only affects a single file. The file type configuration sets the default.




## Text Layout

In EditPad, a "text layout" is a combination of settings that control how text is displayed and how the text cursor navigates through that text. The settings include the font, text direction, text cursor behavior, which characters are word characters, and how the text should be spaced.

You can select two text layouts for each file type. The "default text layout" is used when you first open a file of this file type. It can be a left-to-right or right-to-left text layout. The "layout for opposite text direction" is

used when you turn on Options|Right-to-Left. This layout must be right-to-left if the default is left-to-right, and left-to-right if the default is right-to-left.

You can select previously configured text layouts from the drop-down lists. Click the Edit button next to the list to edit the text layout configurations. The list of text layouts is shared by all file types. If you edit a text layout configuration that is used by multiple file types, the changes apply to all those file types.

See the section describing the Options|Text Layout menu item for a full explanation of the text layout configuration screen.

### Line and Column Numbers

Turn on "show line numbers" to have EditPad display a number before each line in the left margin. In that case, you need to indicate if you want visible lines or physical lines to be numbered. This only makes a difference when word wrapping is on. When numbering visible lines, each line is counted, including lines created by the word wrapping. When numbering physical lines, the lines are numbered as if word wrapping would be off. You can toggle line numbering for a single file with Options|Line Numbers.

Turn on "show column numbers" if EditPad should display a horizontal ruler above the file indicating column numbers. You can toggle this for a single file with Options|Column Numbers. Column numbers are only displayed when using a fixed-width font.

In the box labeled "vertical rulers to the left of specific columns" you can enter a comma-delimited list of column numbers. EditPad will draw a vertical line to the left of the columns you enter. This makes it easier to work with files that use a columnar layout. You can put a + before a number to make that column relative to the previous column. E.g.: 10,+5,+5,37,+7,+12 would be equivalent to 10,15,20,37,44,56.

### Word Wrap

The word wrap setting determines what happens with lines that are too long to fit inside EditPad's window. If you don't want long lines to wrap, they will extend beyond EditPad's window and be partially invisible. A horizontal scroll bar will appear.

You can make long lines wrap either at the edge of EditPad's window, or at a specific line length. When wrapping at the window edge, lines will be automatically re-wrapped when you resize EditPad. When wrapping at a specific line length, horizontal scrolling may still be needed if the specified length doesn't fit inside EditPad's width.

It is usually convenient to wrap at the window edge, as it eliminates the need for horizontal scrolling. You'll probably want to turn it off for files that use a line-based structure such as software source code, so the word wrapping doesn't confuse your view of the lines. Note that EditPad's word wrapping is fully dynamic. Line breaks introduced by the word wrapping are not saved into the file at all, unless you use Convert|Wrapping -> Line Breaks. So there's no risk at data loss when turning on word wrap for line-based files.

If you turn on "wrapped lines should maintain indentation", lines created by the word wrapping will be indented by the same amount as the line they were broken off from. If you turn it off, wrapped lines will start at the first column regardless of the indentation of the original line. Turning on this option is useful when

turning on word wrap for files using nested block structures, such as XML files and various programming languages.

Word wrap can be toggled for individual files with Options|Word Wrap and Options|Indent Wrapped Lines.

## Tabs and Indentation

Normally, pressing the Tab key on the keyboard inserts a special tab character that has a variable width. If you turn on "pressing Tab inserts spaces", then EditPad Pro will insert the equivalent number of spaces, which have a fixed width.

When you turn on "automatically indent new lines when pressing Enter", EditPad will automatically put the same number of spaces and/or tabs at the start of the new line as the previous line starts with. This option can be toggled for individual files with Options|Auto Indent.

"Tab size in spaces" is the maximum width of a tab character expressed in an equivalent number of spaces. A typical value is 8 spaces, which means there will be a tab stop at every 8th column. This option also determines the number of spaces that is inserted if you turn on "pressing Tab inserts spaces".

You can also place tab stops at specific columns by entering a comma-delimited list of column numbers. Pressing Tab will then make the cursor jump to the next tab stop column, either by inserting a single tab character or the number of spaces needed to reach the next tab stop (depending on the "pressing Tab inserts spaces" option). After the last specific column, there will be additional tab stops spaced according to the "tab size in spaces" setting. You can put a + before a number to make that tab stop relative to the previous tab stop. E.g.: 10,+5,+5,37,+7,+12 would be equivalent to 10,15,20,37,44,56.

"Indentation size in spaces" is the number of spaces that the Block|Indent and Block|Outdent will use to indent or outdent the current selection. If the indentation size is an integer multiple of the tab size, and the option to make Tab insert spaces is off, the Indent and Outdent functions will use tab characters.

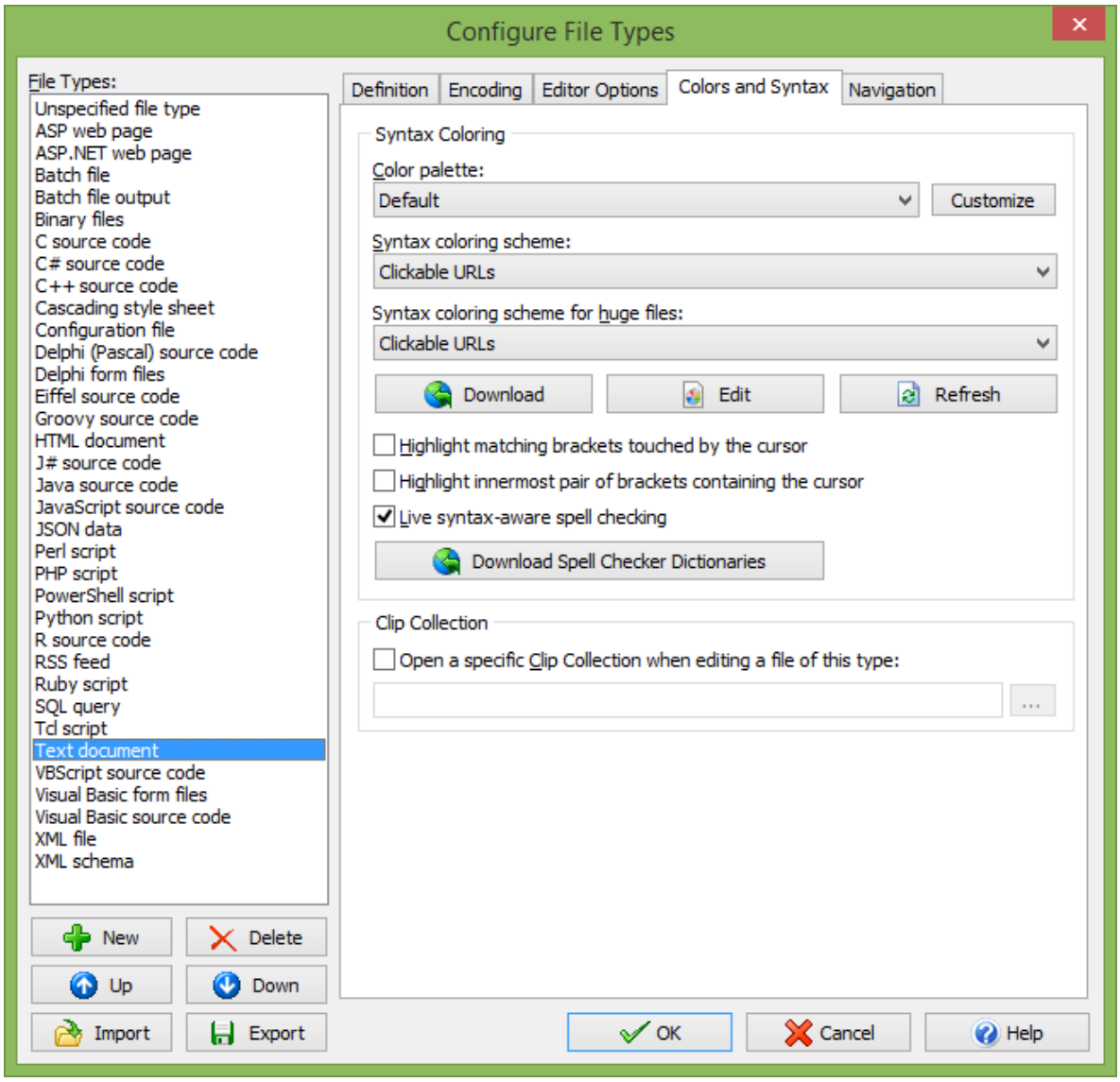
## Visualization

Turn on "show line breaks" if you always want to see a ¶ character at the end of each paragraph, indicating the invisible line break character(s). Note that if line break characters are selected, the paragraph symbol will always appear to show you that they have indeed been selected. You can quickly change this option by picking Options|Paragraph Symbol from the menu.

Select "show spaces and tabs" if you want spaces and tabs to be visualized. Spaces will be indicated by a vertically centered dot. Tabs are indicated by a » character. This option is useful when working with files where extraneous whitespace can lead to problems, or where the difference between tabs and spaces matters. You can quickly toggle it by picking Options|Visualize Spaces from the menu.

# File Type Colors and Syntax

On the Colors and Syntax page in the file types configuration screen, you can configure the syntax coloring and bracket matching text editing aids.




## Syntax Coloring

Syntax coloring highlights different parts of a file in different colors. This makes it easier to edit text files that need to adhere to a certain syntax or formatting, such as programming source code or markup files. The different colors help guide your eyes through the structure of the file.

In EditPad, the actual highlighting is determined by a combination of two settings. The "color palette" associates named colors with actual red-green-blue colors. Select the palette that best matches your color preferences. Click the Customize button next to the drop-down list with color palettes to edit the individual

colors. The list of palettes is shared by all file types. If you edit a palette used by multiple file types, the changes apply to all those file types.

The “syntax coloring scheme” uses the named colors to highlight different parts of the file. EditPad Pro ships with many syntax coloring schemes for a variety of file formats and programming languages. Simply select the one you want to use from the “syntax coloring scheme” drop-down list. Select “none” at the top of the list if you want to disable syntax coloring.

Applying syntax coloring to an entire file takes up too much time and memory for very large files. For files larger than the huge files threshold set in the Open Files Preferences, EditPad Pro automatically disables syntax coloring schemes that require the entire file to be processed. If you selected such a scheme in the “syntax coloring scheme” drop-down list, you can select an alternative scheme in the “syntax coloring scheme for huge files” drop-down list. This second list only shows schemes that do not highlight anything that might span more than one line. This means the scheme only needs to process the visible part of the file. It can instantly apply syntax coloring to files of any size.

Some of the syntax coloring schemes supplied with EditPad Pro come in two versions, one if which is marked (fast). The regular scheme supports the full syntax of the programming language or file format it is intended for, but cannot be used with huge files. The (fast) scheme does not highlight things that span multiple lines, and works with files of any size. For example, the “XML” scheme handles the full XML syntax. The "XML (fast)" scheme highlights everything except comments and CDATA sections that span multiple lines. If a scheme supplied with EditPad Pro is available for huge files and is not marked (fast), that means that the scheme handles the full syntax of the programming language or file format that it is intended for. For example, batch files themselves are line-based, so the scheme for batch files never needs to highlight something that spans multiple lines.

If no coloring scheme is available for the file type you are defining, click the download button. EditPad Pro will then connect to the Internet and allow you to download many custom syntax color schemes created and shared by other EditPad Pro users. To create your own syntax coloring schemes, use the Custom Syntax Coloring Scheme Editor. After editing a scheme or creating new ones, click the Refresh button to make EditPad Pro read in the new and edited schemes.

Turn on “highlight matching brackets touched by the cursor” to highlight the bracket touched by the cursor and that bracket’s corresponding opening or closing bracket. The cursor touches a bracket when it is positioned immediately to the left or right of the bracket. If the bracket consists of multiple characters, the cursor also touches it if it is positioned between two characters that are part of the bracket. If the cursor touches two brackets at the same time that belong to different pairs, the bracket to the right of the cursor is highlighted. If the corresponding bracket is missing, the bracket touched by the cursor is highlighted by itself, in a different color.

Turn on “highlight innermost pair of brackets containing the cursor” to highlight the pair of opening and closing brackets nearest to the text cursor that surround the text cursor. Brackets surround the cursor if the cursor is positioned to the right of the last character in the opening bracket and to the left of the last character in the closing bracket. If the innermost pair of matching brackets contains an unmatched opening bracket to the left of the cursor or an unmatched closing bracket to the right of the cursor, then that unpaired bracket is highlighted alone in a different color and the innermost pair is not highlighted.

If you turn on both bracket highlighting options, then touching brackets are highlighted if the cursor touches a bracket. The innermost pair containing the cursor is highlighted if the cursor does not touch any brackets.

Exactly which brackets are highlighted and how they are paired up depends on the syntax coloring scheme the file type uses. If the syntax coloring scheme does not define any bracket pairs, EditPad matches up nested pairs of `()`, `[]`, and `{}` throughout the file. The provided syntax coloring schemes for programming languages match up these brackets too, depending on how they are used in the programming language. They also match up quotes that are used to delimit strings and characters that delimit multi-line comments. Syntax coloring schemes for markup formats such as HTML and XML match up HTML and XML tags.

EditPad Pro can also apply live spell checking. When you activate the "live syntax-aware spell checking" option, EditPad Pro will mark misspelled words as you type. By default, misspelled words are marked in red and underlined. You can change the look in Colors Preferences. The live spell checking is syntax-aware. This means that the spell checker will work together with the syntax coloring scheme, and only check parts of the file that are likely to contain text in a human language (English). E.g. if you are programming in C# or Java, the spell checker will only check the text in comments and strings. Comments and strings are likely to contain human language, while the rest of the file is in the C# or Java programming language.

The spell checker will only work if you have previously downloaded and installed Just Great Software spell checker dictionaries. Click the Download Spell Checker Dictionaries button to download some or all of the spell check dictionaries free of charge.

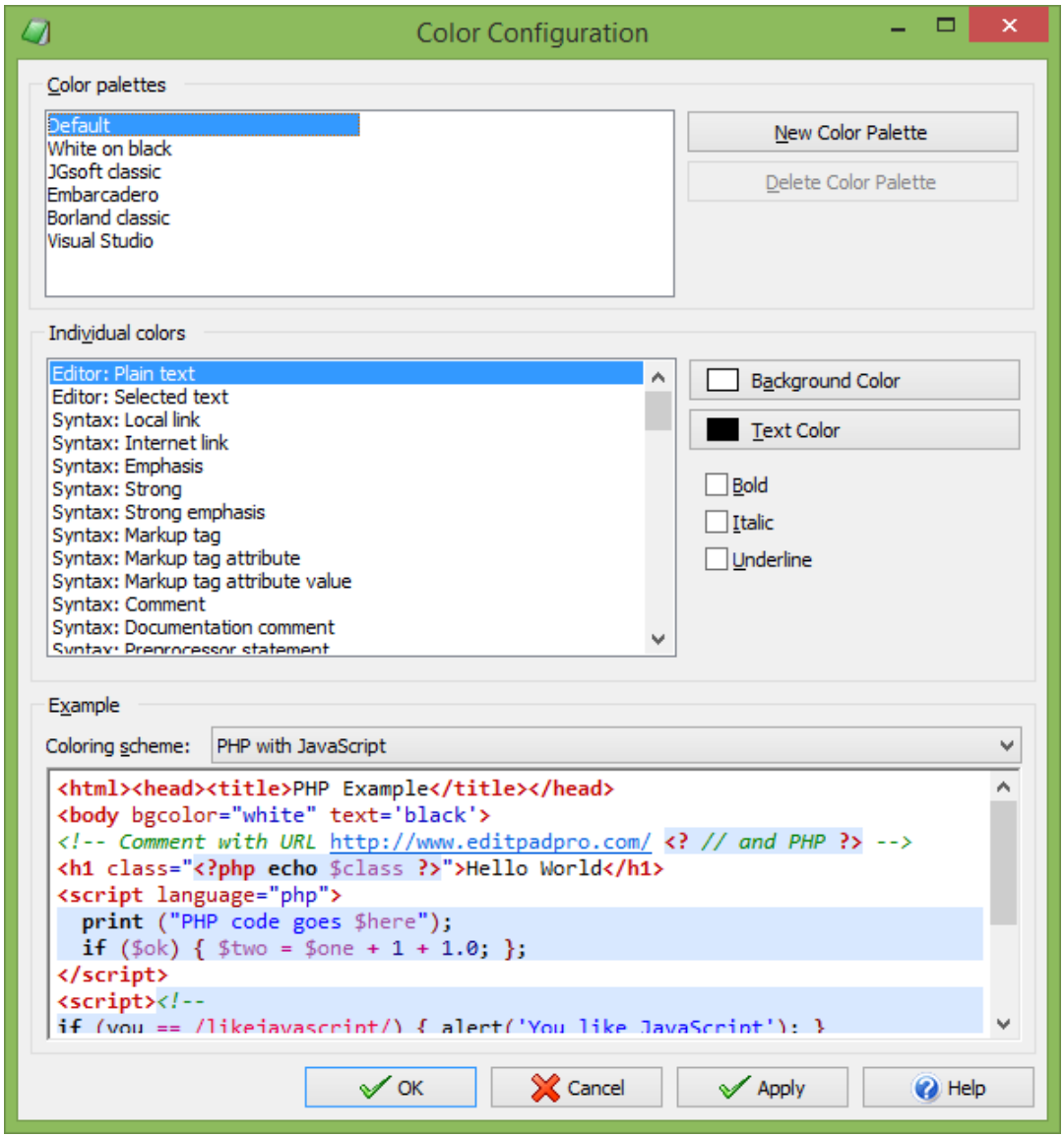
## Clip Collection

You can specify a default Clip Collection to be used for each file type. If you activate a file that has a Clip Collection associated with its file type, that collection is automatically opened. By using different collections for each file type, you always have the appropriate text snippets at your fingertips. E.g. you could associate the HTML file type with a collection with HTML tags, and the Java source code file type with your favorite Java code snippets.

EditPad Pro automatically saves modified collections. If you already have a collection open when activate a file that has a Clip Collection associated with its file type, the collection that was already open is automatically saved before opening the file type's collection.

# Color Palettes

Click the Customize button next to the drop-down list with color palettes in File Types|Colors and Syntax to choose the colors EditPad's editor control will use.




### Color Palettes

At the top of the window , you can select one of the preconfigured color palettes. Selecting one of the preset configurations will change all of the individual colors.

- Default: The default EditPad color palette. This palette has distinct colors for almost all of the items. Choose this configuration if you like very detailed syntax coloring.
- White on black: White text on a black background, with colorful syntax highlighting.
- JGsoft classic: The color scheme used by EditPad Pro 4 and 5, using more saturated colors.

- Embarcadero: Emulates the colors used by recent versions of Delphi and C++Builder.
- Borland classic: Emulates the colors used by old Borland development tools for DOS, with yellow text on a blue background.
- Visual studio: Emulates the colors used by Microsoft's Visual Studio.

### Individual Colors

To change an individual color, simply click on its name. Click on the Background Color or Text Color button to show the color picker. You can pick a common color by clicking on the colored rectangle. You can pick a more specific color by clicking the More button to show the color hexagon. The colors on the outer edge of the hexagon are always fully saturated. The colors nearer to the center of the hexagon are progressively desaturated towards white, gray, or black. Click on the vertical grayscale slider to change the luminance of the colors in the center of the hexagon. If you want to pick a pure grayscale value, click on the horizontal line of small gray hexagons. If you want pure white or black, click on the larger black or white hexagon to the left or right of the gray hexagons. If you want to use a specific RGB value, enter three numbers between 0 and 255 in the edit boxes that are shaded red, green, and blue.

At the top of the color picker there is a button labeled Default. The default color depends on the color item. The default colors for "editor: plain text" are the window text and background colors configured in the Windows Control Panel (as part of the Windows theme or via the advanced appearance settings). The default colors for "editor: selected text" are the selection text and highlight colors specified in the Windows Control Panel. The default colors for all other items are the text and background colors you selected for "editor: plain text" in this palette.

It is possible for multiple colors to apply to the same text. In increasing order of priority, a piece of text may have two levels (subscheme and element) of highlighting by the syntax coloring scheme, may be on a highlighted line (file comparison and active line highlighting), and may be selected. Background and text color are determined separately. EditPad displays each bit of text using the individual color with the highest priority that applies to that text. If that color is set to "default" or is set to the exactly the same color as that for "plain text", EditPad takes the color with the next highest priority. If all colors are set to "default" then the colors for "plain text" are used to display the text. The text color and background color are determined separately. An individual color that uses "default" as its text color and a specific color as its background color gives the appearance of a highlighter. An individual color that uses "default" as its background color and a specific color as its text color gives the appearance of color-coded text with a transparent background. An individual color that has specific text and background colors will appear opaque. An individual color that has both text and background set to "default" is effectively disabled.

If you want to use exactly the same color for two items, and those items are not related via the "default" mechanism, then first select the item that already has the color you want. Click on the Background Color or Text Color button to show the color picker. Observe whether the More button appears flat or depressed. If it is flat, observe which of the squares above the More button has an enclosing bevel. If the More button is depressed, observe whether one of the small hexagons that make up the large color hexagon is surrounded by a thick border. If there is, observe its location as well as the location of the small triangle next to the vertical grayscale bar. If there is not, observe whether one of the black, gray, or white hexagons below the color hexagon is marked. If there is none, make note of the RGB numbers. Making note of the RGB numbers works for all colors, but observing which rectangle or which hexagon is selected is probably faster once you understand the way the color picker works.

After observing the color, click on the item in the list that should have the same color. The color picker automatically closes when you do this. Click on the Background Color or Text Color button again to open

the color picker for the new item. If you observed a color square having a bevel, click that square to select the color. If you observed a small hexagon in the large color hexagon having a thick border, first click on the vertical grayscale bar to move the small triangle to the observed position. Then click on the small hexagon in the large color hexagon to select the color. If you observed a black, gray, or white hexagon having a thick border, click that hexagon. If you noted RGB values, enter those values using the keyboard.

The color hexagon, along with the vertical grayscale slider, makes it easy to pick related colors. The concepts that follow may be easier to understand if you see the large color hexagon as a circle with discrete steps, and the small hexagons that comprise it as positions at a certain radius and angle. Colors with the same distance to the center (same radius) have the same amount of saturation. The colors at the edge are always fully saturated and the color at the center is always fully desaturated. You can change the hue without changing the saturation selecting another color at the same distance (same radius, different angle). You can change the saturation without changing the hue by selecting a color closer to or further away from the center (different radius, same angle). For colors that are not fully saturated, the grayscale slider controls the luminance. To make a color lighter or darker, first click on the grayscale slider to change the luminance. Then click on the same spot in the color hexagon to select the same color with the different luminance.

Syntax elements that use colors of different hues but similar saturation and luminance allow those elements to be distinguished without drawing attention to some over the others. Syntax elements for important structural parts of the file can be given colors with more luminance while syntax elements for less important parts or more common parts (large blocks of text) can be given colors with less luminance to make it easier for the eye to gravitate towards the important elements. This can be further enhanced by using bold text for the most important syntax elements. If the base (plain text) color of the scheme is white, gray, or black, then setting the luminance slider at that level of white, gray, or black allows you to use more saturated colors for important elements and less saturated colors for less important elements, as an alternative technique to using luminance. Color schemes that use very saturated colors appear bolder, while color schemes with (somewhat) desaturated colors (using luminance instead of saturation for highlighting) appear more relaxing.

The colors prefixed with "syntax" are the named colors used by syntax coloring schemes. The descriptions given here are the suggested uses of these colors. The syntax coloring schemes provided with EditPad Pro strictly follow these suggested uses, though most schemes do not use all of the colors. Syntax coloring schemes created by other EditPad Pro users may use these colors to highlight other things.

The colors prefixed with "editor" are used to draw various parts of EditPad's editor control. These colors are not used by syntax coloring schemes. The editor uses these colors even when syntax coloring is disabled.

The colors prefixed with "regex" are used by the Search Panel to apply syntax coloring to regular expressions. Only the full search panel does this. When the search panel is closed, the drop-down lists on the search toolbar do not apply syntax coloring to the regular expression.

- Editor, plain text: The default text colors. EditPad's background is filled with this color, and text that is not syntax colored is drawn in this text color.
- Editor, selected text: Text and highlight colors of the selected text.
- Syntax, local link: Clickable link to a file on the user's computer or local network. Only use this color for scheme elements with an action to open the file.
- Syntax, internet link: Clickable link to a web page, file or email address. Only use this color for scheme elements with an action to open a file or URL, or start an email.
- Syntax, emphasis: Text in a document that will appear in italics when the document is printed or published.
- Syntax, strong: Text in a document that will appear bold when the document is printed or published.

- Syntax, strong emphasis: Text in a document that will appear bold and in italics when the document is printed or published.
- Syntax, markup tag: Opening or closing tag in markup languages.
- Syntax, markup tag attribute: Attribute name in markup languages.
- Syntax, markup tag attribute value: Attribute value in markup languages.
- Syntax, comment: Human-readable text used for information only.
- Syntax, documentation comment: Human-readable text of particular importance, used for information only.
- Syntax, preprocessor statement: Any kind of meta-information, such as compiler and preprocessor directives.
- Syntax, reserved word: Words or character combinations with a specific meaning and specific use, such as keywords in a programming language.
- Syntax, variable name: The name of a variable in a programming language, or a placeholder for a changeable value or macro in a document.
- Syntax, constant name: The name of a constant in a programming language or a fixed placeholder in a document.
- Syntax, function name: The name of a function call in a programming language, or a reference in a document.
- Syntax, character: A single human-readable character (letter).
- Syntax, character string: Human-readable text to be processed by software.
- Syntax, text pattern: A text pattern such as wild cards or a regular expression.
- Syntax, integer number: A whole number.
- Syntax, floating point number: A number with a fractional part and/or an exponent.
- Syntax, date and time constant: Any date or time.
- Syntax, address: Text defining the location of a file or server, such as an IP address. Use this for scheme elements that do not have an action to open or navigate to the address.
- Syntax, operator: Any kind of mathematical operator or other symbol with a specific meaning or effect.
- Syntax, bracket: Round, square, or curly brackets used in expressions, such as parentheses and square brackets in C-style languages.
- Syntax, structural brackets: Round, square, or curly brackets used to group statements or items, such as curly braces in C-style languages.
- Syntax, section header: Heading that starts a new section.
- Syntax, success message: Message in tool output, a log file, or a report indicating successful completion.
- Syntax, hint message: Informative hint message in tool output, a log file, or a report.
- Syntax, warning message: Non-fatal warning message in tool output, a log file, or a report.
- Syntax, error message: Fatal error message in tool output, a log file, or a report.
- Syntax, markup highlight: Highlight color for subschemes that highlight markup in files that do not predominantly consist of markup.
- Syntax, code highlight: Highlight color for subschemes that highlight procedural code in files that do not predominantly consist of procedural code.
- Syntax, fountain highlight 1 to 10: Can be used in specialized schemes to make up to 10 different kinds of items stand out, such as to highlight different kinds of entries in log files.
- Editor, search range: When you start a search with the selection only option turned on, and a match is found, the match will become selected and selection searched through will be highlighted in the search range color.
- Editor, highlight active line: When you turn on the option to highlight the active line, the line containing the text cursor will be highlighted in this color.

- Editor, page break: Color of the horizontal line indicating a page break. (Press Ctrl+Enter to insert one.)
- Editor, line breaks: Color used to draw line break symbols when you've turned on the option to show line break symbols.
- Editor, whitespace: Color used to draw whitespace. The background color is always used if you set it to anything other than "default". The text color is used to draw the space and tab symbols when you turn on the option to visualize spaces.
- Editor, matching brackets: Color used to highlight matching brackets that do not contain any unmatched brackets. This color is also used to highlight the current byte in hexadecimal mode.
- Editor, incorrectly nested brackets: Color used to highlight matching brackets that contain unmatched brackets.
- Editor, unmatched brackets: Color used to highlight brackets that do not have a matching opening or closing bracket.
- Editor, margin and line numbers: Color used for the left and top margins in which line and column numbers are displayed.
- Editor, bookmark icons: Color used to draw the square and number of bookmarks in the left margin.
- Editor, folding icons: Color used to draw text folding buttons and lines.
- Editor, column leader: Color used for vertical rulers marking columns.
- Editor, misspelled word: When using live spell checking, misspelled colors will be indicated using these colors and font style.
- Editor, highlighted search match: Color and font style used for highlighting search matches.
- Editor, compare files, added line: When using Extra|Compare Files, lines present in the second file but not in the first file will be highlighted using this color.
- Editor, compare files, deleted line: When using Extra|Compare Files, lines present in the first file but not in the second file will be highlighted using this color.
- Regex, metacharacter: Any character with a special meaning in regular expressions.
- Regex, Syntax error: Any invalid combination of characters in regular expressions.
- Regex, Comment: A comment in a regular expression.
- Regex, Character class: A character class in a regular expression.
- Regex, Character class metacharacter: Any character with a special meaning inside a character class in a regular expression.
- Regex, Group (nesting level 1 to 5): Any pair of parentheses in a regular expression. Nested parentheses are highlighted in different colors to make it easier to see which opening parenthesis is paired with which closing parenthesis.

### Example

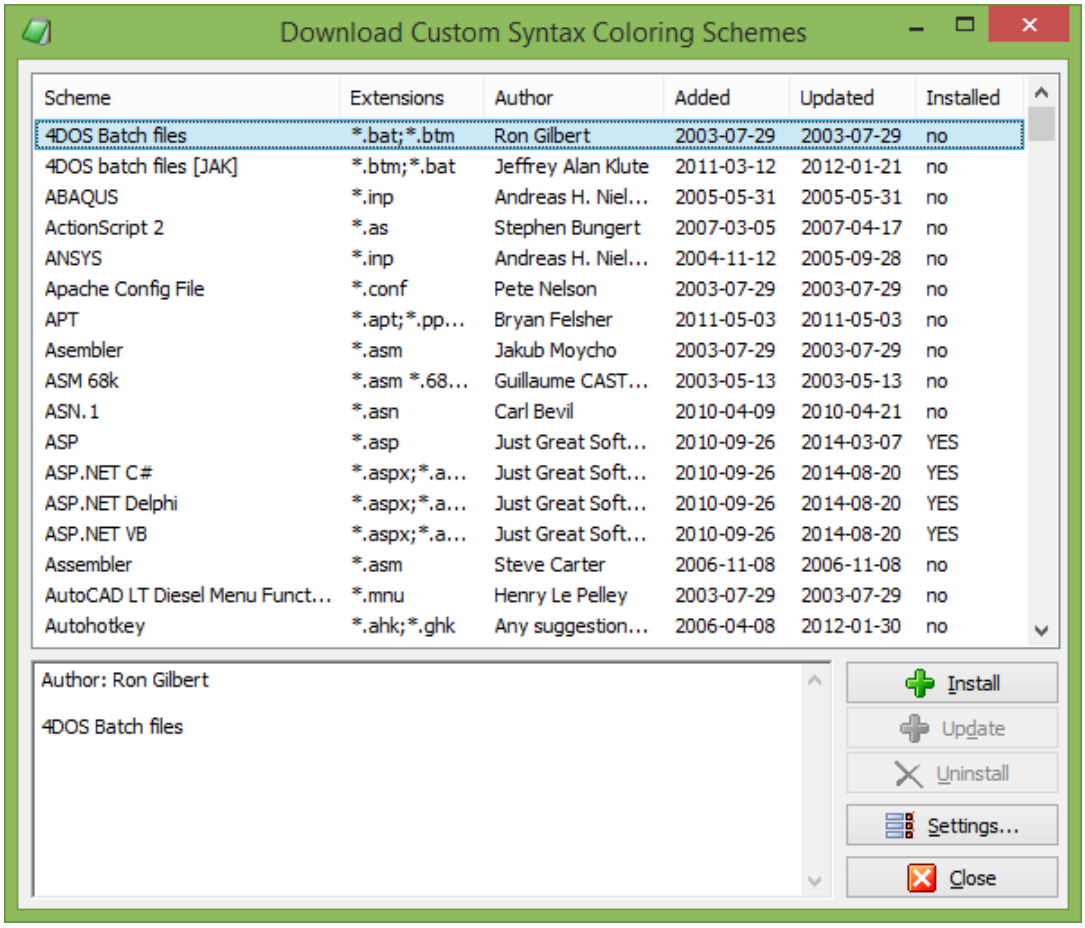
When configuring syntax highlighting colors, you can select one of the available colorings schemes to see an example. Each coloring scheme has its own example text that shows the most important color elements of the scheme. The example does not necessarily show all color elements. You can type in or paste in your own example to test the colors. Your example won't be saved. If you select a different coloring scheme in the drop-down list then the example will be reset to what is stored in the scheme.

## Download Custom Syntax Coloring Schemes

When you click on the button to download custom syntax coloring schemes in the Colors and Syntax file type settings, EditPad Pro will connect to the Internet. It will download a list of custom syntax coloring

schemes that have been created and generously shared by other EditPad Pro users. This list is then displayed in the window shown below.

**Note:** Any schemes you download are *not* automatically put to use. You will need to create or edit a file type to use a scheme that you downloaded.


Download Custom Syntax Coloring Schemes
Scheme
Extensions
Author
Added
Updated
Installed
4DOS Batch files *.bat;*.btm Ron Gilbert 2003-07-29 2003-07-29 no
4DOS batch files [JAK] *.btm;*.bat Jeffrey Alan Klute 2011-03-12 2012-01-21 no
ABAQUS *.inp Andreas H. Niel... 2005-05-31 2005-05-31 no
ActionScript 2 *.as Stephen Bungert 2007-03-05 2007-04-17 no
ANSYS *.inp Andreas H. Niel... 2004-11-12 2005-09-28 no
Apache Config File *.conf Pete Nelson 2003-07-29 2003-07-29 no
APT *.apt;*.pp... Bryan Felsher 2011-05-03 2011-05-03 no
Asembler *.asm Jakub Moycho 2003-07-29 2003-07-29 no
ASM 68k *.asm *.68... Guillaume CAST... 2003-05-13 2003-05-13 no
ASN.1 *.asn Carl Bevil 2010-04-09 2010-04-21 no
ASP *.asp Just Great Soft... 2010-09-26 2014-03-07 YES
ASP.NET C# *.aspx;*.a... Just Great Soft... 2010-09-26 2014-08-20 YES
ASP.NET Delphi *.aspx;*.a... Just Great Soft... 2010-09-26 2014-08-20 YES
ASP.NET VB *.aspx;*.a... Just Great Soft... 2010-09-26 2014-08-20 YES
Assembler *.asm Steve Carter 2006-11-08 2006-11-08 no
AutoCAD LT Diesel Menu Funct... *.mnu Henry Le Pelley 2003-07-29 2003-07-29 no
Autohotkey *.ahk;*.ghk Any suggestion... 2006-04-08 2012-01-30 no
Author: Ron Gilbert
4DOS Batch files
Install
Update
Uninstall
Settings...
Close

The list presents you the following information about the available schemes:

- **Scheme:** The name of the scheme. It is possible that more than one scheme with the same name is available. If so, you can install all of them or only one, as you prefer.
- **Extensions:** The extensions typically used for the files that the scheme provides syntax coloring for. They are only shown for your information. You can apply any scheme to any file type, as you see fit.
- **Author:** The author of the scheme. If you have comments or suggestions about a particular coloring scheme, you should contact this person.
- **Added:** The date the scheme was first added to the list of available schemes.
- **Updated:** The date the scheme was last updated.
- **Installed:** Indicates whether this scheme is installed on your computer or not.

If you click the **Install** button, EditPad Pro will immediately download the scheme and save it into the `%APPDATA%\JGsoft\EditPad Pro 7` folder.

Click the **Update** button to download a scheme that you already installed again.

If you click the **Uninstall** button, EditPad Pro will delete the scheme from your computer when you close the window for downloading custom syntax coloring schemes.

If you are behind a proxy server, and EditPad Pro is unable to detect your proxy settings, you can change them by clicking the **Settings** button.

## Syntax Coloring Scheme Editor

With the Syntax Coloring Scheme Editor, you can create your own syntax coloring schemes for use with EditPad Pro. You can also edit the schemes included with EditPad Pro, and those you've downloaded. The syntax coloring schemes files have a .jgcscs extension. You can't edit them with any program except the Syntax Coloring Scheme Editor. You can download the scheme editor at http://www.editpadpro.com/cscs.html. The editor is a free download for all licensed EditPad Pro users. Full documentation is included.

The syntax coloring schemes included with EditPad Pro are stored in the folder where you have EditPad Pro installed, typically c:\Program Files\JGsoft\EditPadPro 7. On Windows Vista and Windows 7, the default security settings do not allow normal users to modify files under c:\Program Files. Therefore, schemes that you download and schemes that you edit are saved under your Windows user profile, typically c:\Users\yourname\AppData\Roaming\JGsoft\EditPad Pro 7. If a scheme with the same file name exists in both the user profile folder and the program files folder, only the scheme in the user profile folder will be available in EditPad. In the Colors and Syntax section in the file type configuration, schemes loaded from your Windows user profile are marked with (user).

## Using Custom Syntax Coloring Schemes

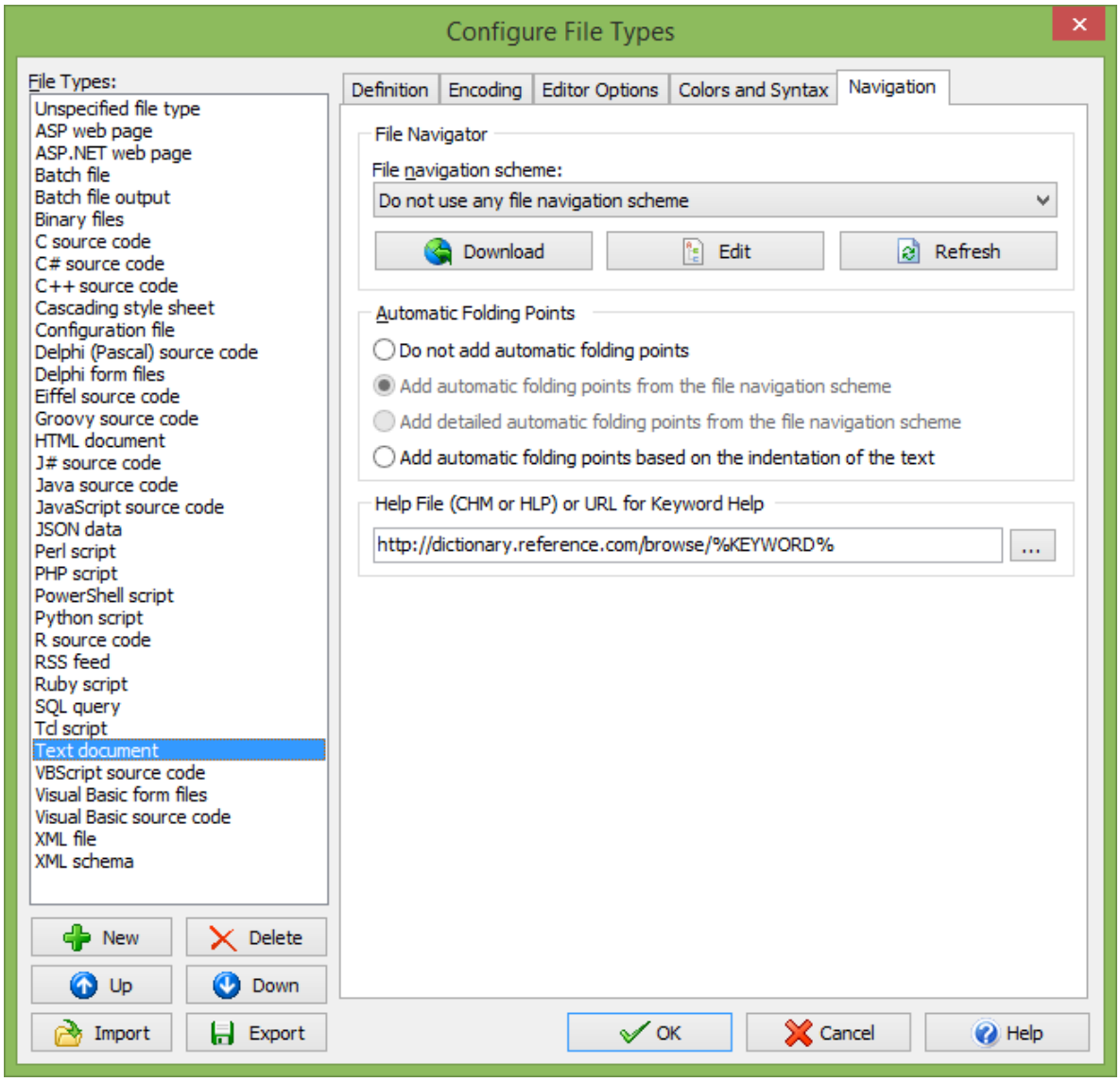
After downloading a custom syntax coloring scheme, it is *not* automatically put to use.

All the schemes that you downloaded, will be listed in the Syntax Coloring drop-down list in the Colors and Syntax file type settings. They will appear in alphabetic order among the schemes that ship with EditPad Pro. This drop-down list is shown in the screen shot below.

To use one of the schemes, create or edit a file type, and select the coloring scheme you want from the Syntax Coloring drop-down list.

# File Type Navigation

On the Navigation page in the file types configuration screen, you can configure the file navigation and line folding text editing aids.




### File Navigator

You can access the File Navigator by selecting View|File Navigator in the menu. The File Navigator displays the structure of the file in a collapsible tree. By clicking on items in the tree, you can quickly navigate to various parts of the file.

The File Navigator requires a file navigation scheme to do its job. You can select a predefined scheme from the file "navigation scheme" drop-down list.

If no file navigation scheme is available for the file type you are defining, click the download button. EditPad Pro will then connect to the Internet and allow you to download file navigation schemes created and shared by other EditPad Pro users. To create your own file navigation schemes, use the File Navigation Scheme Editor. After editing a scheme or creating new ones, click the Refresh button to make EditPad Pro read in the new and edited schemes.

### Automatic Folding Points

Automatic folding points appear as small squares in the left margin, with a vertical line extending down from the square to indicate the range. These allow you to quickly fold logical parts of the text to get a better overview of the overall structure. When EditPad Pro adds automatic folding points, any unused (i.e. expanded) folding points you created with Fold|Fold and Fold|Unfold are removed.

EditPad Pro can obtain automatic folding points from two sources. Many, but not all, file navigation schemes also define foldable ranges when they associate different parts of the file with various nodes in the file navigation tree. These ranges usually follow the syntax of the file. Some file navigation schemes mark some of their foldable ranges as being "detailed". For example, schemes for C-style languages add regular folding ranges for classes and functions, and detailed folding ranges for all other pairs of curly braces. If you select to add automatic folding points from the file navigation scheme, only the regular folding ranges will appear in EditPad Pro. If you select to add detailed automatic folding points from the file navigation scheme, regular and detailed folding ranges will appear in EditPad Pro. Both types of folding ranges behave in exactly the same way. The only difference is that the detailed ranges don't show up at all if you don't choose that option.

Alternatively, EditPad Pro can use a file's indentation as the basis for folding points. Whenever a line is followed by one or more lines that are indented further than itself, that line will become a folding point. Its range will stop before the next line with the same or a smaller indentation than the foldable line. EditPad Pro will nest folding points this way up to three levels deep. Further levels down will only be added if the foldable range is longer than half the number of lines that EditPad Pro can display at a time.

Note that when enabling automatic folding points, you can still use the Fold|Fold to fold different blocks of text. They just won't persist when you unfold them and edit the file.

### Help File or URL for Keyword Help

When you press F1 in EditPad Pro, normally EditPad Pro's help file appears. But if you enable keyword help for a file type, and the main editor has keyboard focus when you press F1, then EditPad Pro can show help for the file type you're working with EditPad Pro itself. If you selected a block of text that does not span more than one line before pressing F1, EditPad Pro shows help for the selected text. Otherwise, it shows help for the word under the cursor.

If you specify the full path to a HLP or CHM file, EditPad Pro automatically looks up the selected text or the word in the cursor in the index of the HLP or CHM file. You can click the (...) button to select a HLP or CHM file on your computer. If you are running Windows Vista or Windows 7 and you specify a HLP file, you may be prompted to download the WinHelp viewer from Microsoft if you haven't done so already.

You can also specify the URL to a web page. If you do, use the %KEYWORD% placeholder in the URL. EditPad Pro will substitute this placeholder with the selected text or the word under the cursor. E.g. if you want to start a Google search when you press F1, set the URL to

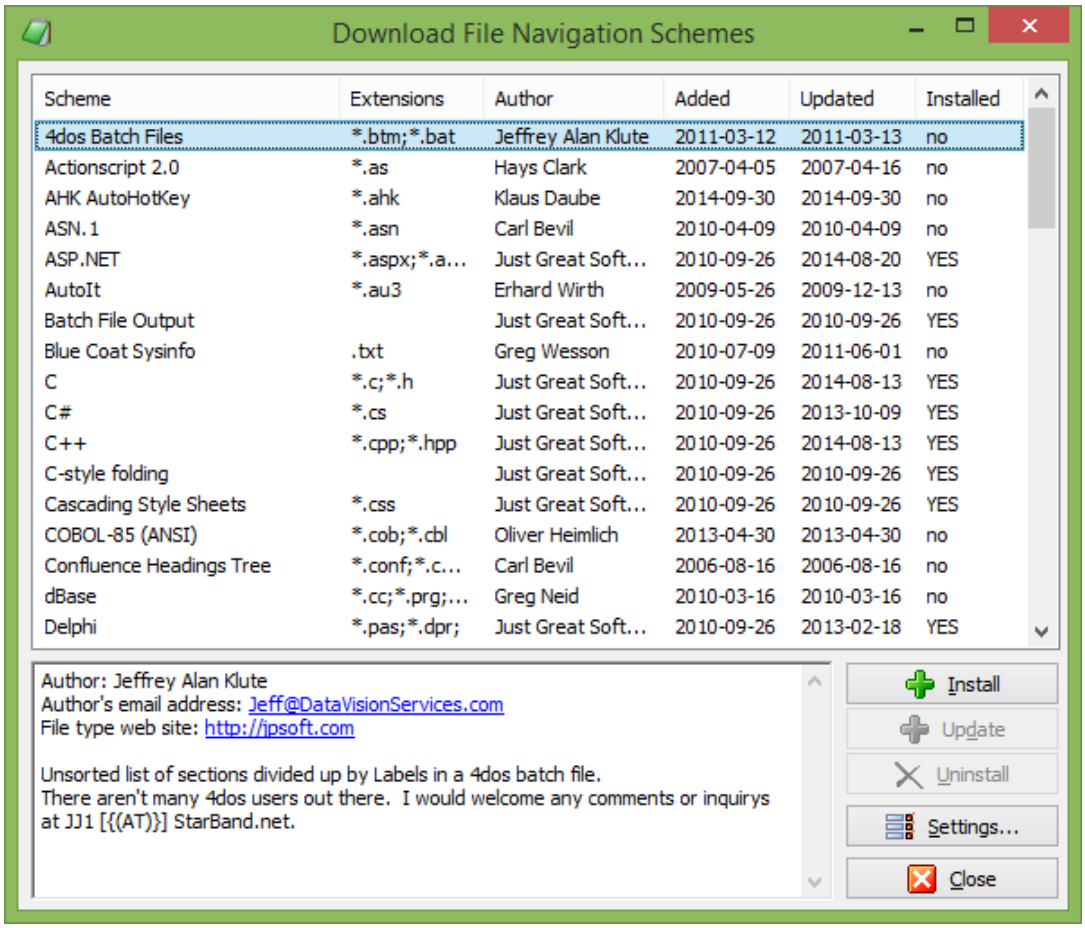
`http://www.google.com/search?q=%KEYWORD%`. EditPad Pro automatically URL-encodes any special characters in the selected text.

Finally, you can specify the path to any document or a command line to any application, including command line parameters. You can use the %KEYWORD% placeholder anywhere on the command line. EditPad Pro will substitute the placeholder without giving special treatment to any characters in the selection or the word under the cursor.

## Download File Navigation Schemes

When you click on the button to download file navigation schemes in the Navigation file type settings, EditPad Pro will connect to the Internet. It will download a list of custom syntax coloring schemes that have been created and generously shared by other EditPad Pro users. This list is then displayed in the window shown below.

**Note:** Any schemes you download are *not* automatically put to use. You will need to create or edit a file type to use a scheme that you downloaded.


Download File Navigation Schemes
Scheme
Extensions
Author
Added
Updated
Installed
4dos Batch Files *.btm;*.bat Jeffrey Alan Klute 2011-03-12 2011-03-13 no
Actionscript 2.0 *.as Hays Clark 2007-04-05 2007-04-16 no
AHK AutoHotKey *.ahk Klaus Daube 2014-09-30 2014-09-30 no
ASN.1 *.asn Carl Bevil 2010-04-09 2010-04-09 no
ASP.NET *.aspx;*.a... Just Great Soft... 2010-09-26 2014-08-20 YES
AutoIt *.au3 Erhard Wirth 2009-05-26 2009-12-13 no
Batch File Output Just Great Soft... 2010-09-26 2010-09-26 YES
Blue Coat Sysinfo .txt Greg Wesson 2010-07-09 2011-06-01 no
C *.c;*.h Just Great Soft... 2010-09-26 2014-08-13 YES
C# *.cs Just Great Soft... 2010-09-26 2013-10-09 YES
C++ *.cpp;*.hpp Just Great Soft... 2010-09-26 2014-08-13 YES
C-style folding Just Great Soft... 2010-09-26 2010-09-26 YES
Cascading Style Sheets *.css Just Great Soft... 2010-09-26 2010-09-26 YES
COBOL-85 (ANSI) *.cob;*.cbl Oliver Heimlich 2013-04-30 2013-04-30 no
Confluence Headings Tree *.conf;*.c... Carl Bevil 2006-08-16 2006-08-16 no
dBase *.cc;*.prg;... Greg Neid 2010-03-16 2010-03-16 no
Delphi *.pas;*.dpr; Just Great Soft... 2010-09-26 2013-02-18 YES
Author: Jeffrey Alan Klute
Author's email address: Jeff@DataVisionServices.com
File type web site: http://jpsoft.com
Unsorted list of sections divided up by Labels in a 4dos batch file.
There aren't many 4dos users out there. I would welcome any comments or inquirys
at JJ1 [{(AT)}] StarBand.net.
Install
Update
Uninstall
Settings...
Close

The list presents you the following information about the available schemes:

- **Scheme:** The name of the scheme. It is possible that more than one scheme with the same name is available. If so, you can install all of them or only one, as you prefer.
- **Extensions:** The extensions typically used for the files that the scheme provides syntax coloring for. They are only shown for your information. You can apply any scheme to any file type, as you see fit.
- **Author:** The author of the scheme. If you have comments or suggestions about a particular file navigation scheme, you should contact this person.
- **Added:** The date the scheme was first added to the list of available schemes.
- **Updated:** The date the scheme was last updated.
- **Installed:** Indicates whether this scheme is installed on your computer or not.

If you click the **Install** button, EditPad Pro will immediately download the scheme and save it into the `%APPDATA%\JGsoft\EditPad Pro 7` folder.

Click the **Update** button to download a scheme that you already installed again.

If you click the **Uninstall** button, EditPad Pro will delete the scheme from your computer when you close the window for downloading custom syntax coloring schemes.

If you are behind a proxy server, and EditPad Pro is unable to detect your proxy settings, you can change them by clicking the **Settings** button.

## Using File Navigation Schemes

After downloading a file navigation scheme, it is *not* automatically put to use.

All the schemes that you downloaded, will be listed in the File Navigation Scheme drop-down list in the Navigation file type settings. They will appear in alphabetic order among the schemes that ship with EditPad Pro. This drop-down list is shown in the screen shot below.

To use one of the schemes, create or edit a file type, and select the coloring scheme you want from the File Navigation Scheme drop-down list.

## File Navigation Scheme Editor

With the File Navigation Scheme Editor, you can create your own file navigation schemes for use with EditPad Pro. You can also edit the schemes included with EditPad Pro, and those you've downloaded. The file navigation schemes files have a .jgfns extension. You can't edit them with any program except the File Navigation Scheme Editor. You can download the scheme editor at http://www.editpadpro.com/fns.html. The editor is a free download for all licensed EditPad Pro users. Full documentation is included.

The file navigation schemes included with EditPad Pro are stored in the folder where you have EditPad Pro installed, typically c:\Program Files\JGsoft\EditPadPro 7. On Windows Vista and Windows 7, the default security settings do not allow normal users to modify files under c:\Program Files. Therefore, schemes that you download and schemes that you edit are saved under your Windows user profile, typically c:\Users\yourname\AppData\Roaming\JGsoft\EditPad Pro 7. If a scheme with the same file name exists in both the user profile folder and the program files folder, only the scheme in the user profile folder will be

available in EditPad. In the Navigation section in the file type configuration, schemes loaded from your Windows user profile are marked with (user).

# 15. Options|Preferences

When you pick Options|Preferences from the menu, the EditPad Preferences screen appears. Since there are a lot of things you can configure, the available settings are split into several tabbed pages.

EditPad is very configurable, and the number of choices may seem a bit overwhelming. Fortunately, if a certain option seems meaningless to you, you can simply leave it in its default state. The default settings have been carefully chosen so that EditPad is fully functional even if you do not make any changes in the Preferences screen at all. In its default configuration, EditPad will work like most Windows text editors.

All the options you can set in the Preferences screen are global settings, affecting EditPad's overall behavior. Many editing and display settings are available through the file type configuration, allowing you to use different settings for different kinds of files.

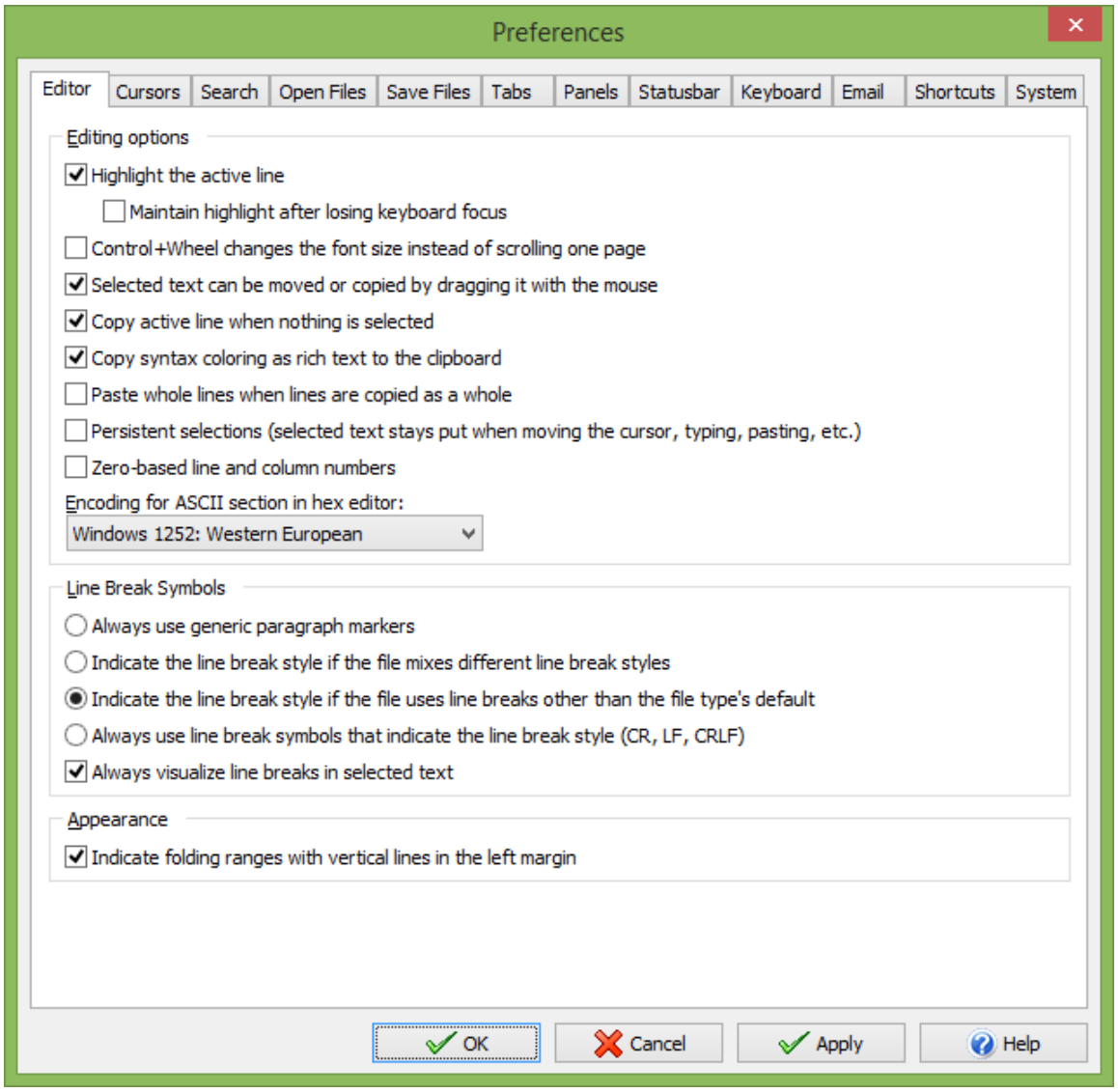
The available pages are:

- Editor
- Cursors
- Open Files
- Save Files
- Tabs
- Panels
- Statusbar
- Keyboard
- Email
- Shortcuts
- System

The main menu, all toolbars, and various right-click menus can be configured by right-clicking on the main menu or on any toolbar and selecting Customize.

# Editor Preferences

On the Editor tab of the Preferences you can set the options that affect basic editing tasks that are not file type specific.


Preferences
Editor
Cursors
Search
Open Files
Save Files
Tabs
Panels
Statusbar
Keyboard
Email
Shortcuts
System
Editing options
Highlight the active line
Maintain highlight after losing keyboard focus
Control+Wheel changes the font size instead of scrolling one page
Selected text can be moved or copied by dragging it with the mouse
Copy active line when nothing is selected
Copy syntax coloring as rich text to the clipboard
Paste whole lines when lines are copied as a whole
Persistent selections (selected text stays put when moving the cursor, typing, pasting, etc.)
Zero-based line and column numbers
Encoding for ASCII section in hex editor:
Windows 1252: Western European
Line Break Symbols
Always use generic paragraph markers
Indicate the line break style if the file mixes different line break styles
Indicate the line break style if the file uses line breaks other than the file type's default
Always use line break symbols that indicate the line break style (CR, LF, CRLF)
Always visualize line breaks in selected text
Appearance
Indicate folding ranges with vertical lines in the left margin
OK
Cancel
Apply
Help

## Editing Options

Highlighting the active line makes it easier to keep track of where you are in the file, particularly when switching between EditPad and other applications. The active line is the line the text cursor is on. Just like the text cursor itself, the active line is only highlighted when the editor has keyboard focus. If you want it to be highlighted permanently, turn on “maintain highlight after losing keyboard focus” too. You can configure the color of the active line in the color palette for each file type. Click on "editor: highlight active line" in the list. Click the Background Color button to change the highlight color.

In EditPad, rotating the mouse wheel scrolls the active file 3 lines up or down. Holding down the Ctrl key while rotating the mouse wheel scrolls the active file one screen up or down. Essentially, holding down the Ctrl key speeds up scrolling with the mouse wheel. In many other applications, Ctrl+Wheel zooms in our out. As a plain text editor, EditPad Pro does not have the ability to zoom. But EditPad Pro can mimic zooming by increasing or decreasing the font size of the active file. Turn on "Control+Wheel changes the font size instead of scrolling one page" if you prefer to change the font rather than to scroll quickly when using Ctrl+Wheel.

By default, “selected text can be moved or copied by dragging it with the mouse” is turned on. This allows drag-and-drop editing within EditPad. It also allows you to drag text from EditPad into other applications. People with limited dexterity using a mouse may find themselves accidentally moving text when trying to select text. Turning off this option prevents that.

In EditPad, the Edit|Cut and Edit|Copy are always enabled by default. If no text is selected, these commands cut or copy the active line. This makes allows you to quickly cut and copy whole lines, as it removes the need to select them. You can turn off the option “copy active line when nothing is selected” if you want EditPad to disable the Cut and Copy commands when no text is selected, as most Windows applications do.

If you copy text to the clipboard while editing a file using syntax coloring, then EditPad places both plain text and rich text on the clipboard. When you paste the rich text version into a word processor, it will be pasted with the same font and colors you were using in EditPad. The rich text version is only generated when another application requests it. When copying and pasting within EditPad or between EditPad and another plain text editor, no time or memory is wasted to generate the RTF. If you don’t want word processors to paste rich text, either use the “paste as plain text” or “paste unformatted text” command in your word processor, or tell EditPad Pro not to copy it by turning off the option “copy syntax coloring as rich text to the clipboard”.

Normally, when you paste text, that text is inserted at the position of the text cursor, regardless of where the cursor is placed and which text is being pasted. If you often copy and paste whole lines of text, you may want to turn on “paste whole lines when lines are copied as a whole”. A line is copied as a whole if all the text on that line and the line break at the end of the line are selected when you copy them. If the option to paste whole lines is on, and you copy a whole line in EditPad, and then paste it, the line is pasted before the line that the cursor is on, as if the cursor was positioned at the start of the line. This allows you to quickly copy and paste whole lines without worrying about the horizontal position of the text cursor. This option also affects the Block|Move and Block|Duplicate commands in the same way. If you move or duplicate a whole line, the line is moved or copied as if the cursor was on the start of the line that it is on. This option only affects whole lines copied in EditPad. When pasting text copied from another application, EditPad cannot determine whether it was a complete line or not.

By default, selections in EditPad Pro behave like in most other Windows applications. When you move the text cursor, by pressing a key on the keyboard or clicking somewhere with the mouse, the selection will disappear. When typing in some text or pasting from the clipboard, the selected text will be replaced by the text you typed in or pasted. If you turn on “persistent selections”, moving the text cursor will not cause the selection to disappear. Also, typing in text or pasting text will not delete the selected text. The typed or pasted text will be inserted at the position of the text cursor, whether that is outside the selection, at the edge of the selection, or inside the selection. If it is inside the selection, the selection will be expanded to include both the text originally selected and the newly entered text.

Turn on "zero-based line and column numbers" if you want EditPad Pro to start counting lines and columns from zero instead of one. This option affects the status bar, as well as Options|Line Numbers and Options|Column Numbers.

You can change the encoding for the ASCII section in the hex editor if you work with binary files that also store text in a particular code page. This setting changes the way bytes are translated into characters in the right hand section of the text editor, and how characters that you type in are translated into bytes. You can only select 8-bit code pages, as the hex editor displays one character per byte.

## Line Break Symbols

You can show or hide line break symbols with the Options|Visualize Line Breaks command. You can set the default in the editor options for each file type. These two determine whether line breaks are visualized or not. The Line Break Symbols section in the Preferences determines how they are visualized.

- Always use paragraph markers: Always visualize line breaks using the ¶ symbol.
- Indicate the line break style if the file mixes different line break styles: Visualize line breaks using the ¶ symbol if the line breaks in the file are all Windows, all UNIX, or all Mac line breaks. If the file uses a mixture of line breaks, Windows line breaks are indicated as $^{C_R}{}_{L_F}$, UNIX line breaks are indicated as ${}_{L_F}$, and Mac line breaks are indicated as $^{C_R}$.
- Indicate the line break style if the file uses line breaks other than the file type's default: Visualize line breaks using the ¶ symbol if all the line breaks in the file uses the default line break style set in the encoding settings for the file type. If the file has at least one line break that does not use the file type's default line break style, then Windows line breaks are indicated as $^{C_R}{}_{L_F}$, UNIX line breaks are indicated as ${}_{L_F}$, and Mac line breaks are indicated as $^{C_R}$. This option is the default. It makes it easy to see whether all line breaks are as expected (¶ symbol used for all line breaks) or whether there's at least one odd line break (line-break specific symbols used for all line breaks).
- Always use line break symbols that indicate the line break style: Windows line breaks are indicated as $^{C_R}{}_{L_F}$, UNIX line breaks are indicated as ${}_{L_F}$, and Mac line breaks are indicated as $^{C_R}$.

If you choose to show the generic ¶ symbol then you'll need to rely on the status bar to determine the line break style used by the file.

By default, EditPad shows line break symbols for all line breaks that are part of a selected block, even when you've turned off the option to visualize line breaks. EditPad Lite, which does not have any options for visualizing line breaks, does this too. This makes it easy to tell the difference between selecting a line of text and selecting a line of text including the line break at the end of the line. In EditPad Pro, you can disable this by turning off "always visualize line breaks in selected text". If you do this, there is no visual difference between selecting a line with or without the line break that terminates it.

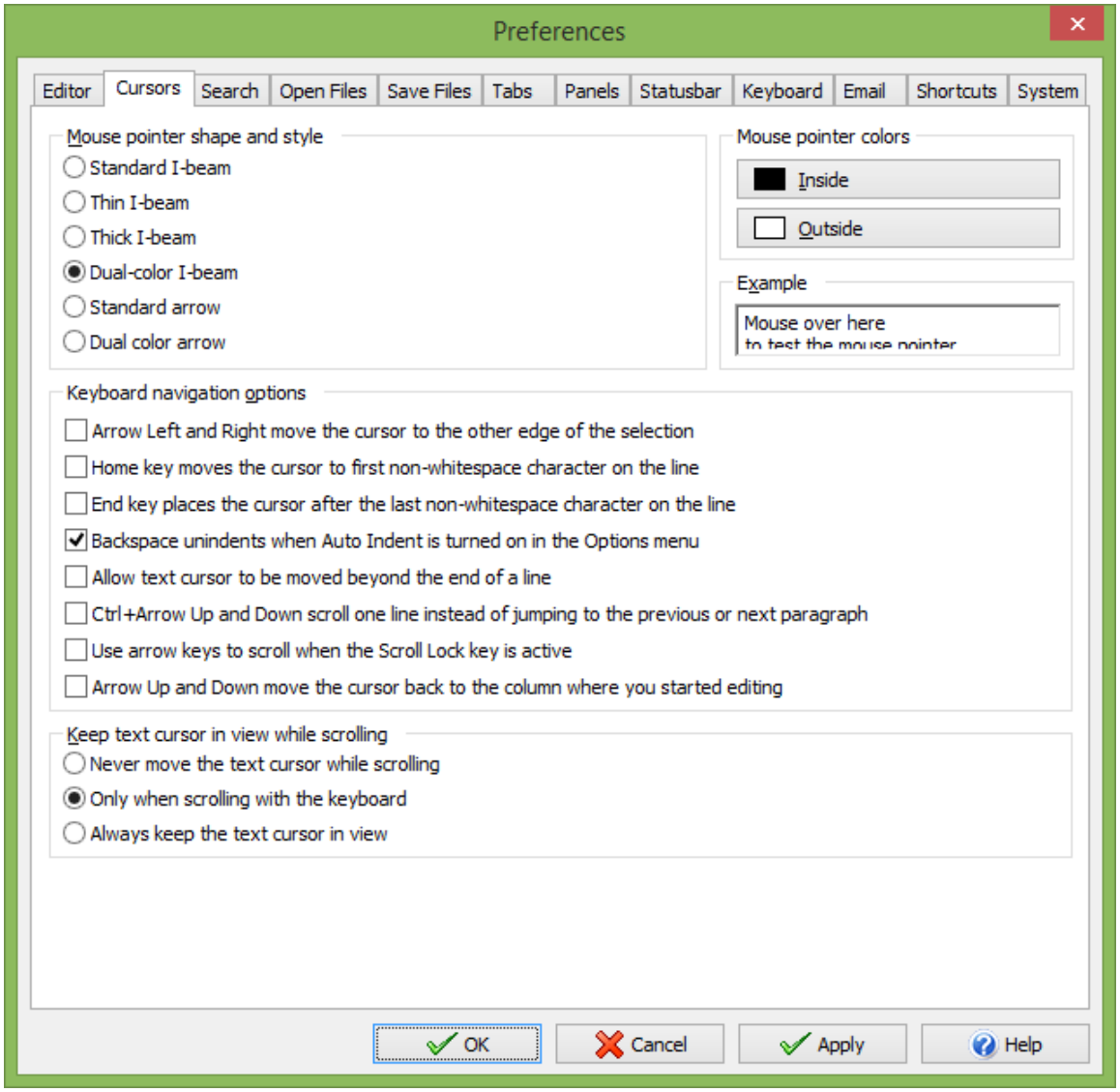
## Appearance

When you use the Fold|Fold command to fold a range of lines, EditPad Pro indicates those with a "plus" button in the left margin. When you use Fold|Unfold, the "plus" will change into a "minus", and a vertical line will appear next to the range of lines. Automatic folding points also appear this way. If you find the vertical lines distracting, you can disable them. Then only the "plus" or "minus" buttons will appear. You may prefer this style if you only work with automatic folding ranges that span logical blocks that are obvious from the file's content.

# Cursors Preferences

On the Cursors tab in the Preferences screen, you can configure the appearance of EditPad's mouse pointer. You can also configure how EditPad responds to the navigational keys on the keyboard.

The appearance of the text cursor (blinking vertical bar) cannot be set in the Preferences. That can be set in the text layout configuration because EditPad's text cursor can indicate text direction.


Preferences
Editor
Cursors
Search
Open Files
Save Files
Tabs
Panels
Statusbar
Keyboard
Email
Shortcuts
System
Mouse pointer shape and style
Standard I-beam
Thin I-beam
Thick I-beam
Dual-color I-beam
Standard arrow
Dual color arrow
Mouse pointer colors
Inside
Outside
Example
Mouse over here
to test the mouse pointer
Keyboard navigation options
Arrow Left and Right move the cursor to the other edge of the selection
Home key moves the cursor to first non-whitespace character on the line
End key places the cursor after the last non-whitespace character on the line
Backspace unindents when Auto Indent is turned on in the Options menu
Allow text cursor to be moved beyond the end of a line
Ctrl+Arrow Up and Down scroll one line instead of jumping to the previous or next paragraph
Use arrow keys to scroll when the Scroll Lock key is active
Arrow Up and Down move the cursor back to the column where you started editing
Keep text cursor in view while scrolling
Never move the text cursor while scrolling
Only when scrolling with the keyboard
Always keep the text cursor in view
OK
Cancel
Apply
Help

### Mouse Pointer Shape and Style

EditPad Pro allows you to select the mouse pointer shape and style. "Standard I-beam" is the standard mouse pointer for text editing controls, typically an I-shaped beam. "Thin I-beam", "thick I-beam" and "dual-color I-beam" are custom EditPad cursors for which you can pick your own colors. By choosing colors that contrast well with the background color you've chosen for the editor, you can make the mouse pointer highly

visible. "Standard arrow" is the regular Windows mouse pointer. "Dual color arrow" is another custom EditPad cursor in the shape of an arrow.

EditPad's custom pointers may not work in all situations. E.g. some remote desktop software cannot handle them. The mouse pointer may either be incorrectly displayed, or simply be invisible. In that case, simply select one of the standard pointers, which will always work.

## Keyboard Navigation Options

In some editors, like Notepad, pressing Arrow Left or Right moves the cursor one character left or right and clears the selection if there is one. In other editors, like Wordpad, pressing Arrow Left or Right while there is a selection puts the cursor on the left or right edge of the selection and then clears the selection. By default, EditPad behaves like Notepad. If you turn on "arrow Left and Right move the cursor to the other edge of the selection" then EditPad behaves like Wordpad. This option is only available when the "persistent selections" option in the Editor Preferences is off. When selections are persistent, the Left and Right arrow keys always move the cursor one character and never clear the selection.

By default, pressing the Home key moves the text cursor to the very start of the line it is on. This is how the Home key works traditionally. In some modern editors, pressing the Home key moves it to the left of the first non-whitespace character of the line it is on. Pressing the Home key a second time moves it to the very start of the line. You can make the Home key work this way in EditPad Pro by turning on "Home key moves cursor to first non-whitespace character on the line".

Similarly, pressing the End key normally moves the text cursor to the very end of the line. Turning on "End key places the cursor after the last non-whitespace character on the line" makes the first press of the End key move the cursor after the last non-whitespace character, requiring a second press of the End key to go to the very end of the file.

When you turn on Auto Indent in the Options menu or in the editor options in the file type configuration then pressing Enter to create a new line automatically duplicates all whitespace at the start of the previous line onto the new line. When Auto Indent is on, there are two ways in which the Backspace key can delete this whitespace at the start of a line. If you turn on "Backspace unindents" then pressing Backspace deletes as many whitespace characters as needed to line up the cursor with the previous indentation level. That is the amount of spaces used to indent the first line before the line the cursor is on that has less indentation than the horizontal position of the text cursor when you press the Backspace key. So if you start with a blank file, type 3 spaces, type text, press Enter (which indents the new line by 3 spaces), type 3 more spaces, type text, press Enter (which indents the 3rd line by 6 spaces) and then press Backspace, then 3 spaces are deleted to line up the cursor with the first line's indentation. Pressing backspace again deletes 3 more spaces. If you turn off "backspace unindents", then pressing Backspace always deletes one character. This option has no effect when Auto Indent is off. Without auto indent, backspace always deletes one character.

If you turn on "allow text cursor to be moved beyond the end of a line", you can position the text cursor after the last character of the line. If you press the right arrow key when the cursor is at the end of the line, it will move one position to the right. If you click with the mouse beyond the end of the line, the cursor will be placed where you clicked. If you start typing when the cursor is beyond the end of the line, EditPad will automatically fill up the line with spaces up to the position where you started typing. When you do not allow the cursor to be moved beyond the end of a line, pressing the right arrow key when the cursor is at the end of a line will move it to the start of the next line. When you click the mouse beyond the end of the line, the cursor will be placed after the last character on the line.

You can choose what happens when you press Control+Arrow Up or Down on the keyboard. By default, this will scroll the text one line up or down without moving the text cursor, as if you had clicked on the up or down arrow button on the scroll bar, or as if you had rotated the mouse wheel. Ctrl+Arrow Up and Down work this way in most programmer's text editors. If you prefer to use the mouse rather than the keyboard for scrolling, you can configure the Ctrl+Arrow Up and Down keys to make the text cursor jump to the next or previous paragraph. Ctrl+Arrow Up and Down work this way in some word processors like Microsoft Word.

If you turn on "use arrow keys when the Scroll Lock key is active", then you can push the Scroll Lock key on the keyboard to scroll with the arrow keys. Pressing any of the arrow keys will then scroll the view instead of moving the text cursor. The Home and End keys will scroll to the top and the end of the file. Press Scroll Lock again to restore the normal arrow key behavior. If this option is on, then EditPad Pro ignores the state of the Scroll Lock key.

In Windows text editors, pressing the Arrow Up or Down keys on the keyboard keeps the text cursor on the same column. E.g. if you put the cursor on line 1, column 1 of an existing text file, type "abc", and then press Arrow Down, then the cursor will be on line 2, column 4. If you type "abc" on the 2nd line, it will be "stair-stepped" relative to the "abc" on the first line. If you turn on the option "Arrow Up and Down move the cursor back to the column where you started editing", then in the previous example, pressing Arrow Down will move the cursor to line 2, column 1. The horizontal movement caused by typing "abc" is undone by pressing Arrow Down. If you type "abc" on the second line, it will appear just below the "abc" on the first line. The result is that turning on this option makes it much easier to edit data arranged in columns.

### Keep Text Cursor in View While Scrolling

"Keep text cursor in view while scrolling" determines what happens to the text cursor when you scroll the text. Normally, the text cursor will keep its position relative to the text when you scroll the text. This may cause the text cursor to be scrolled off-screen. Depending on your text editing habits, this may be desirable or undesirable. If you allow the text cursor to be scrolled off-screen, it will keep its position relative to the text. After scrolling, you can immediately continue typing at the same position. When you do so, the text will be scrolled automatically to make the cursor visible again. If you choose to keep the text cursor visible, it will be moved to the topmost or bottommost visible line when scrolling would otherwise have made it invisible. This way, you can always see exactly where the text cursor is pointing to.

By default, EditPad Pro will keep the text cursor visible when you scroll with the keyboard (by pressing Ctrl+Arrow Up/Down or Ctrl+Page Up/Down), but not when you scroll with the scroll bars or by rotating the mouse wheel. This is how most text editors for programmers work. EditPad Pro gives you the choice. Most general-purpose text editors do not allow keyboard scrolling at all.

## Mouse Actions

Below you can find a list of action you can carry out in the editor using the mouse.

Dragging means to move the mouse before releasing the mouse button you pressed. If you move the mouse pointer to the edge of the editor space while dragging, the text will start to scroll automatically.

Modifier keys like shift or control must be pressed before pressing the mouse button and kept depressed until the mouse button is released.

Left click: Moves the text cursor to the spot where you clicked. Any text that was previously selected becomes unselected, unless you enabled persistent selections and clicked outside the selection.

Shift+Left click: Moves the text cursor and expands or shrinks the selection. If there is no selection, the text between the old and new cursor positions becomes selected. If you click outside of the selection, the selection plus the text between the selection and the new cursor position becomes selected. If you click inside the selection, the new selection is the text between the original start of the selection and the new cursor position.

Ctrl+Left click: Moves the text cursor to the spot where you clicked. Selects the line that you clicked on entirely, unless there already is a selection and you enabled persistent selections.

Ctrl+Shift+Left click: Moves the text cursor and expands the selection just like Shift+Left click does. In addition, if the selection starts and/or ends in the middle of a line, the selection will be expanded to include the partially selected lines entirely.

Left click+drag: When clicking outside the selection, a new selection is created from the point where you press the mouse button until the point where you release it. When clicking inside the selection, the selected text deleted and inserted again at the spot (outside the selection) where you release the mouse button.

Shift+Left click+drag: Expands or shrinks the selection like Shift+Left click, but then the text cursor is moved and the selection adjusted until you release the mouse button.

If you press Alt while changing the selection with any of the above left click methods, the selection becomes rectangular. If Block|Rectangular Selections is active, pressing Alt makes the selection linear.

Left double click: Moves the text cursor to the spot where you clicked. If you double-clicked on a word, the word becomes selected. If you double-click whitespace, all adjacent whitespace becomes selected. If you double-click anything else, all characters up to the preceding and following word become selected. All occurrences of the newly selected text will be highlighted if you turned on the option to make double-clicking highlight all occurrences of a word in the Search Preferences.

Shift+Left double click: Moves the text cursor and expands or shrinks the selection just like Shift+Left single click does. In addition, if the selection starts and/or ends in the middle of a word, the selection will be expanded to include the words at the start and/or end entirely.

Left double click+drag: A new selection is created from the point where you double-clicked until the point where you release it. If the selection starts and/or ends in the middle of a word, the selection will be expanded to include the words at the start and/or end entirely.
Shift+Double click+drag Expands or shrinks the selection like Shift+Left double click, but then the text cursor is moved and the selection adjusted until you release the mouse button. Words at the edge of the adjusted selection will be selected entirely.

Left triple click: Moves the text cursor to the spot where you clicked. Selects the line that you clicked on entirely (even when selections are persistent).

Shift+Left triple click: Moves the text cursor and expands or shrinks the selection just like Shift+Left single click does. In addition, if the selection starts and/or ends in the middle of a line, the selection will be expanded to include the partially selected lines entirely.

Left triple click+drag: A new selection is created from the point where you triple-clicked until the point where you release it. If the selection starts and/or ends in the middle of a line, the selection will be expanded to include the partially selected lines entirely.

Shift+Triple click+drag: Expands or shrinks the selection like Shift+Left triple click, but then the text cursor is moved and the selection adjusted until you release the mouse button. Lines part of the adjusted selection will be selected entirely.

Rotate wheel: Scrolls the text a single line up or down.

Shift+Wheel: Moves the text cursor a line up or down, like pressing the up or down arrow keys on the keyboard.

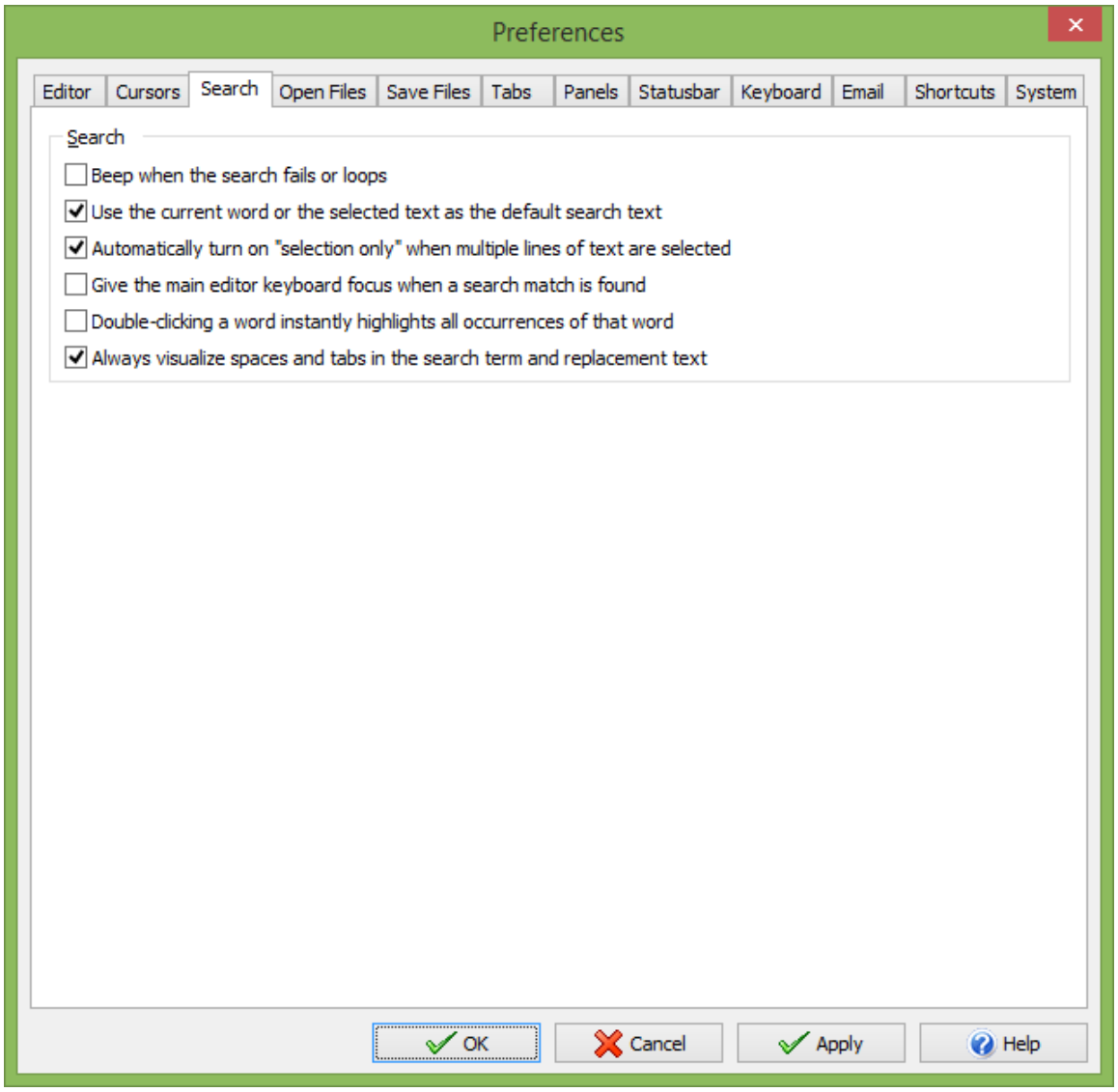
Ctrl+Wheel: Scrolls the text an entire screen up or down.

Shift+Ctrl+Wheel: Moves the text cursor a screen up or down, like pressing page up or down on the keyboard.

Scrolling the text with the wheel, the text cursor keeps its position relative to the text. This may cause the text cursor to be scrolled off screen and become invisible. If you do not like this, select "always keep the text cursor in view" in the Cursor Preferences. Then, scrolling with the wheel will also move the text cursor if needed to keep it visible.

# Search Preferences

On the Search tab of the Preferences you can set the options that affect basic editing tasks that are not file type specific.




When a search action fails, EditPad Pro does nothing except flash the icon of the button you clicked on the Search panel. If this is not obvious enough for you, you can turn on the option "beep when the search fails or loops". A standard "error" beep will then sound when a search fails. If the "loop automatically" search option is on and EditPad Pro finds a match after restarting from the beginning, then a standard "notification" beep will sound.

By default, EditPad will automatically copy the selected text into the search panel when you prepare to search, if the selection does not span multiple lines. If no text was selected, EditPad will copy the word under the text cursor. If the selection spans more than one line when you open the search panel, EditPad will automatically turn on the "selection only" option. You can disable this behavior by turning off the option to

use the selected text as the default search text. Then, opening the search panel will show it unchanged. The last search term you entered will still be there, and the state of the "selection only" option will be unchanged.

If there is a selection that spans multiple lines, then by default EditPad will turn on the Selection Only search option when you prepare to search. You can disable this by turning off the option to automatically turn on "selection only" when multiple lines of text are selected.

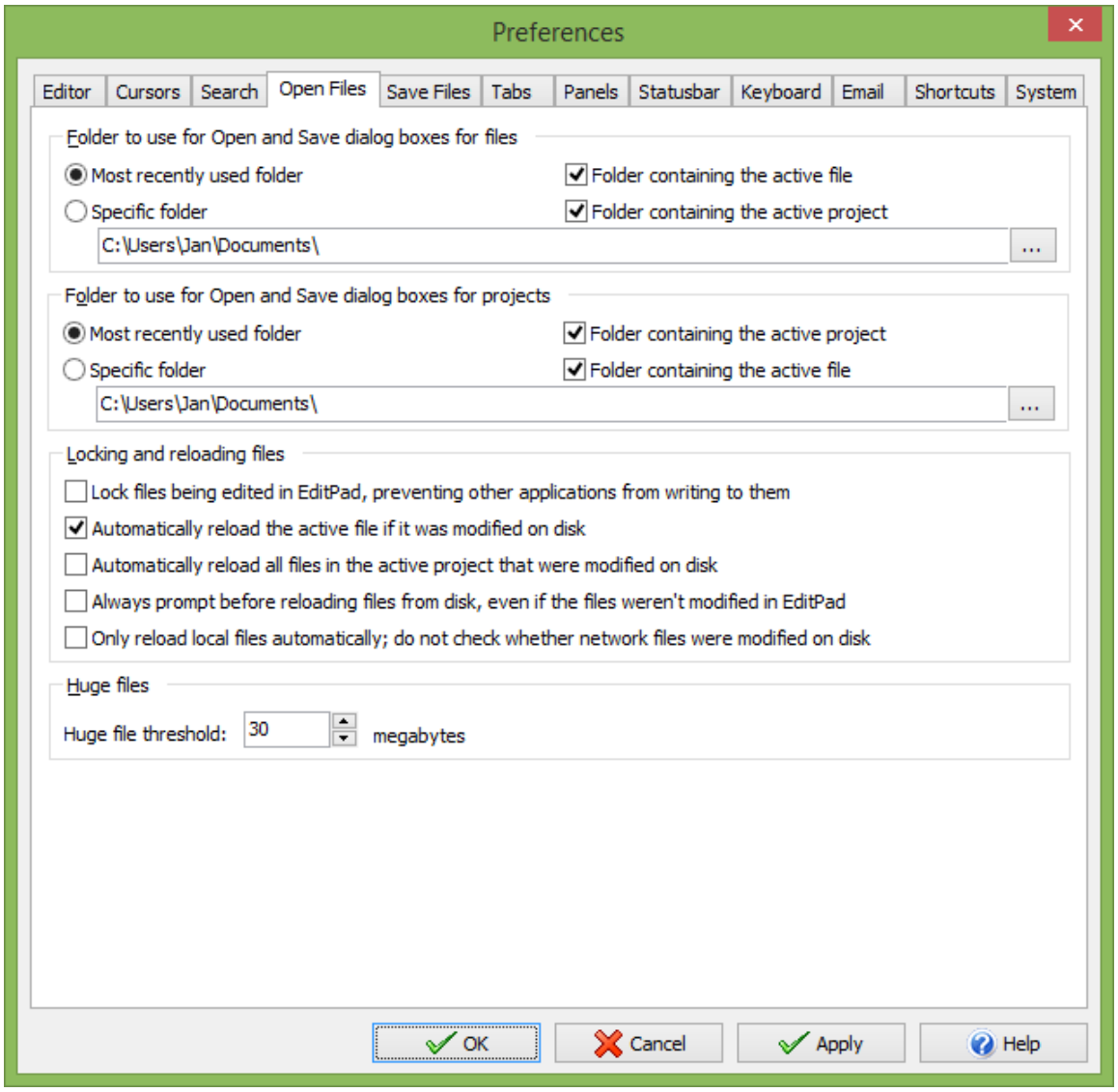
Turn on "give the main editor keyboard focus when a search match is found" if you want to be able to immediately edit the search match after doing a Find First or Find Next. You can press F3 on the keyboard to continue to the next search match, even when the editor has keyboard focus and even after the search panel has been closed. However, if you prefer to be able to continue editing the text you're searching for rather than the text found, turn this option off.

Turn on "double-clicking a word instantly highlights all occurrences of that word" if you want to be able to invoke the Search|Instant Highlight command by double-clicking. Double-clicking still selects the word you've double-clicked on, regardless of whether this option is on or off.

Unintended spaces and line breaks may cause EditPad to apparently not find search terms that do occur in the text, when the spurious spaces and line breaks don't occur in the text. To avoid this, you can turn on the option to always visualize spaces and tabs in the search panel. When this option is off, the search panels only visualizes these characters when you've turned on the option to visualize spaces and line breaks in the file you're editing.

# Open Files Preferences

On the Open Files tab in the Preferences screen, you can configure how EditPad should handle files when opening them.




### Folder to Use for Open and Save Dialog Boxes

You can select if the file selection dialog boxes that appear when you want to open or save a file. First, you can choose which folder is the default when you do not have any file or project open in EditPad, or when the active file or project is untitled. The default folder can either be the last folder you opened a file from or saved a file into, or it can be a specific folder such as your "My Documents" folder.

In addition to specifying the default folder, you can choose if the folder containing the active project or file should be used instead of the default. If you turn on the active file option, the folder containing the active file

will be used for opening or saving a file when you have a file open. If you turn on the active project option, and the active file option is off or the active file is untitled, the folder containing the active project will be used instead. The default folder is only used if you turned off both options, or the active file and project are both untitled.

You can also set a default folder for opening and saving projects in the same way.

## Locking and Reloading Files

By default, EditPad does not keep a lock on files. This means that other applications or users can modify files that you have open in EditPad. If you don't want this to happen, turn on "lock files being edited in EditPad, preventing other applications from writing to them". The option whether or not to lock files only applies to files smaller than the huge file threshold (see below). Files larger than the huge file threshold are always locked. EditPad Pro doesn't read those files into memory entirely, so it needs to keep access to the original file on disk.

When not locking files, EditPad can automatically reload files that have been modified on disk by another application or user. If you turn on this option, EditPad will check whether a file's modification date has changed each time you switch between files in EditPad, and each time you switch between EditPad and another application.

By default, when you switch from another application to EditPad, EditPad only checks whether the active file was modified on disk. If you have multiple files open in EditPad that were all modified on disk, you'll only be prompted to reload the active file. When you switch to the other files, you'll be prompted for each of them at the moment you switch to them. If you turn on "automatically reload all files in the active project that were modified on disk", then EditPad Pro checks all files in the active project whenever you switch from another application to EditPad. If multiple files were modified, you'll be prompted for all of them at the same time. You'll be able to choose for each file whether you want to reload it or not. Regardless of whether this option is on or off, when you switch files within EditPad, EditPad check whether the file that you're switching to needs to be reloaded.

When automatically reloading, EditPad will prompt when you've modified the file in EditPad, but not when you haven't modified the file in EditPad. Turn on "always prompt before reloading files from disk" to make EditPad prompt to reload the file even when you haven't modified it in EditPad.

When editing files over a slow network connection, checking whether a file was modified on disk may cause a brief delay when switching between files in EditPad or when switching from another application to EditPad. If you experience this, turn on "only reload files automatically; do not check whether network files were modified on disk". Then the options to automatically reload will only apply to files stored on your own computer. Modern storage devices allow EditPad to do this check instantly. EditPad won't check files stored on another PC or server that you're accessing via the Windows network, so a slow network doesn't slow down your work in EditPad.

## Huge Files

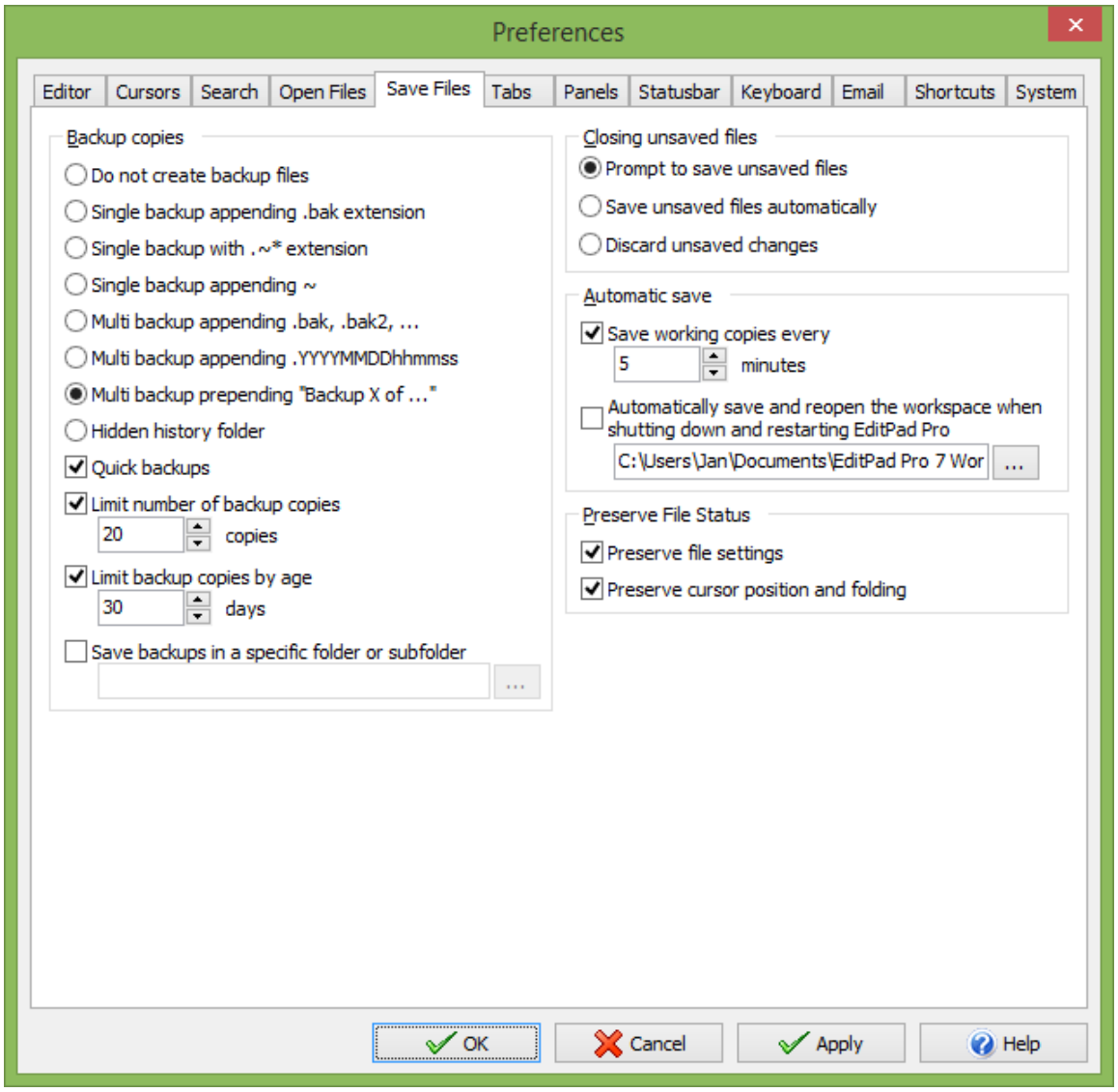
EditPad Pro is capable of handing files of almost any size, including files larger than 4 gigabytes. However, loading such large files entirely into memory would quickly exhaust the available memory of most computers, slowing down the system to a crawl.

Therefore, you can set an upper limit on the size of files that EditPad Pro will read into memory entirely. You can choose any threshold between 10 megabytes and 10% of the total amount of RAM in your PC. On 32-bit Windows, the maximum threshold is 200 MB, even if your PC has more than 2 GB of RAM. Files larger than your chosen threshold will be swapped out to disk. This slightly slows down EditPad, because reading from disk is slower than reading from memory. But it makes sure EditPad does not use too much RAM so your computer keeps running smoothly. To be able to read files as needed, EditPad Pro needs to keep a lock on those files. This means that EditPad Pro will lock files larger than the huge file threshold, even when you've turned off the option to keep a lock on open files.

EditPad Pro also uses the huge files threshold to disable certain processing that would take too much (CPU) time on very large files, so that you don't have to wait on EditPad or have your laptop's battery drained by EditPad unnecessarily. Syntax coloring is disabled for huge files, unless you've selected a "fast" syntax coloring scheme in the file type configuration. Fast syntax coloring schemes are schemes that don't require the whole file to be processed. File navigation schemes are also disabled. The File Navigator will remain blank and automatic folding points will not appear. When opening the file or toggling between text and hexadecimal mode, EditPad will put the cursor at the start of the file rather than at the previous position, so that you don't have to wait for EditPad to scan line breaks up to the previous cursor position.

# Save Files Preferences

On the Save Files tab in the Preferences screen, you can configure how EditPad should handle files when saving them.




## Backup Copies

To make sure you never lose any data in case you change your mind after saving a file, you should turn on one of the options to create backup copies. The "single backup" options will keep one backup copy of each file in the same folder as the original. The "multi backup" options will keep multiple backup copies in the same folder. The "hidden history" option will create a hidden "__history" folder below each folder, and put the backup copies there. The "multi backup" and "hidden history" options work best with EditPad Pro's File History. The File History lets you easily compare, delete and revert to backup copies.

By default, EditPad uses a quick method to make backups. It moves the original file into its backup location, and then writes the file you're saving as a new file. This works perfectly when editing files on your Windows PC or on a Windows server, but may cause issues if you're editing a file on other systems, such as a Linux server that makes itself visible on the Windows network using Samba. When EditPad moves the file into a backup location and writes a new file, the newly saved file may not have the same UNIX file permissions of the original file. To prevent this, turn off "quick backups". Then EditPad makes backups by making a copy of the original file. Then the original file is modified with the new text you're saving.

EditPad can automatically limit the number of backup copies for each file to a certain number. When the number of backup copies is exceeded, EditPad will automatically delete the oldest ones. EditPad can also limit backup copies by age. When you save the file, backup copies older than the specified age will be automatically deleted. You should turn on at least one of the backup limitations when choosing "multi backup" or "hidden history" to prevent an ever-increasing number of backup files from taking up too much disk space.

If you enable both limits, they are enforced simultaneously. EditPad will never keep more backups of a single file than the maximum limit, even if that means EditPad has to delete files that are younger than your age limit. It will also never keep backups older than your age limit, even if the maximum number of backup files has not yet been reached.

The limits are only enforced when EditPad actually creates a backup copy. If you set the age limit to 1 month, edit a file regularly for 3 days (creating many backups), and then don't edit the file for a year, those year-old backups will remain. The year-old backups will only be deleted if you save the file again, leaving a single backup copy (with the contents of the file that you overwrote after one year).

Normally, EditPad saves backups in the same folder as the original files, or in a hidden "__history" subfolder in that folder. If you'd like all backups to be saved into a particular folder, turn on "save backups in a specific folder or subfolder" and specify the full path to the folder where you would like to keep your backup copies. If you'd like the backups to be created in a subfolder of the folder that contains the original file and choose the name of that subfolder, turn on the "save backups in a specific folder or subfolder" option and type in the name of the subfolder.

If you save backups on a separate drive and that drive is running low on disk space, you may want to use a file manager to delete all files older than a certain age and/or larger than a certain size. Since EditPad does no bookkeeping of its backup files, deleting them using another application causes no issues. In EditPad Pro you can also use the File History to delete all backup copies of the current file or project.

## Closing Unsaved Files

The "closing unsaved files" option gives You have three choices as to what EditPad should do when you instruct it to close a file to which you have made changes that were not yet saved to disk:

- Show a warning message and give you one last opportunity to decide if the file should be saved or not. This is the default since most other software behaves this way too.
- EditPad can automatically save the file without asking you first. This is the best way to make sure you will not lose any work. For safety, you should also select one of the options to make multiple backup copies.
- The last option will make EditPad close the file without warning. Any unsaved work will be lost, so use with caution.

## Automatic Save

If you want EditPad Pro to automatically save your work regularly, select to save working copies. EditPad Pro will then save a copy of all modified files every couple of minutes. When working on document.txt, a file Working copy of document.txt will be saved into the same folder as document.txt. When you save document.txt or close document.txt without saving, the working copy will be automatically deleted.

If the document is not properly closed, because of a power loss or software crash, the working copy will not be deleted. When you open document.txt next time, EditPad Pro will automatically open both document.txt and its working copy. You can then choose which file you want to keep and which you want to delete, possibly after using Extra|Compare Files to assist with your choice.

Note: If you have both "document.txt“ and ”working copy of document.txt" open in EditPad Pro, and you start editing the former, the latter will be automatically overwritten after a few minutes. So if you want to keep the working copy, you should use File|Save As before editing the base document.

Working copies of unsaved files are saved in the “specific folder” you have specified in the “folder to open for open and save dialog boxes for files” section on the Open Files tab in the Preferences, even when you have “most recently used folder” selected there. If you restart EditPad Pro after a system crash, the working copies of unsaved files are opened automatically when you start EditPad.

If you want to continue working with the same set of files the next time you start EditPad Pro, turn on “automatically save and reopen workspace”. You can also choose the file into which EditPad Pro should have its workspace. The workspace will contain a list of project and files that you have open. The next time you start EditPad Pro, those projects and files will be automatically reopened. If you restart EditPad Pro after a system crash, it will open the workspace along with all working copies of all files in the workspace.

## Preserve File Status

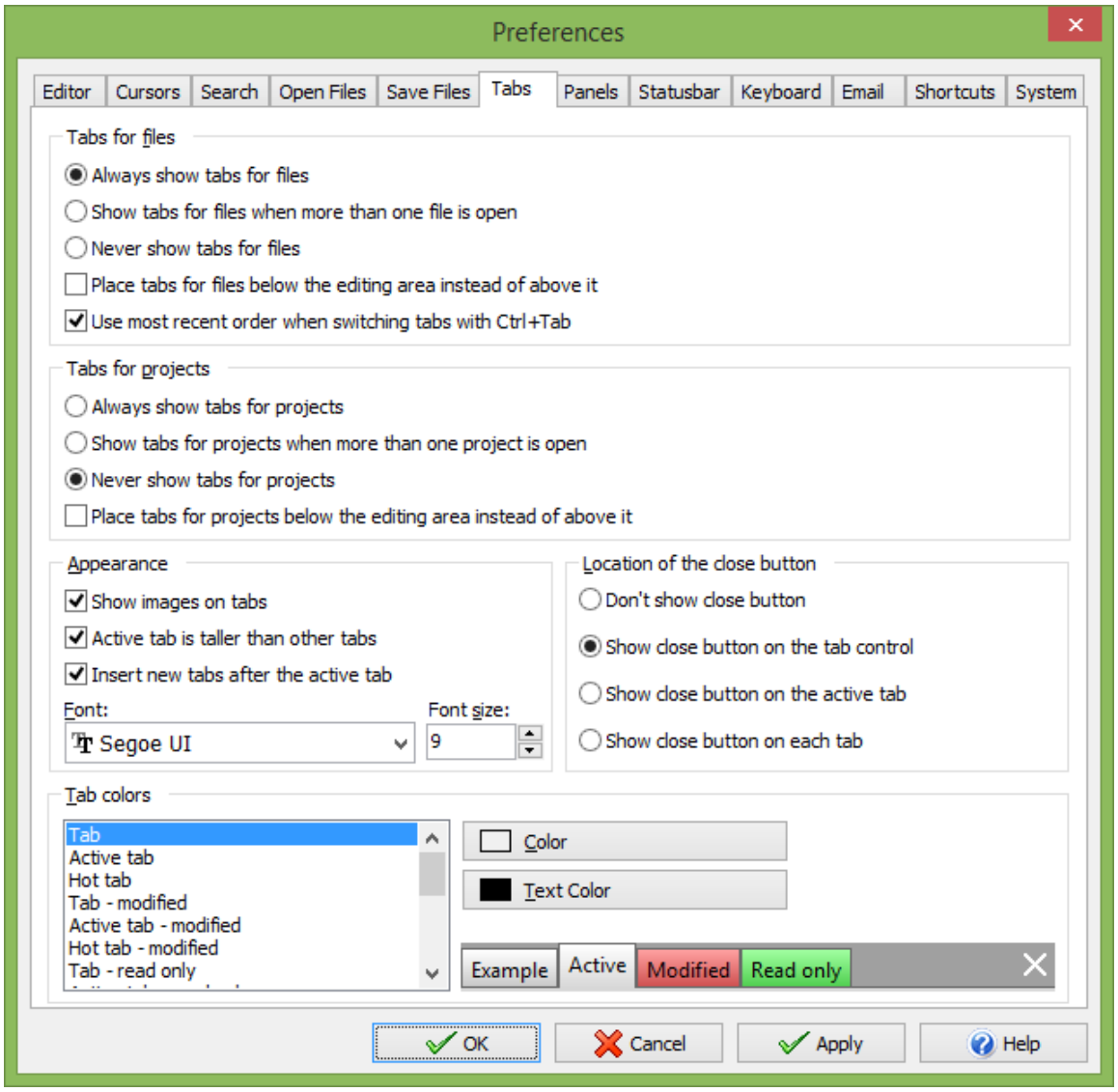
By default, EditPad Pro remembers a whole bunch of status information for the files that you edit. This information is stored for all the files listed in the File|Open and File|Favorites submenus. It is also saved into `.epp` project files whenever you save a project. This way, each file will appear the next time you open it in EditPad Pro the way it did last time.

With “preserve file settings” turned on, EditPad Pro will store the settings from the Options menu that you’ve changed from their defaults in the file type configuration. With “preserve cursor position and folding” turned on, EditPad Pro will remember the position of the text cursor in the file, which part of the file was selected, bookmarks and folding.

Opening large files may take a bit longer when preserving the cursor position, as EditPad Pro then has to scan the file for line breaks up to the cursor position before it can show the file. Because of this, EditPad Pro won’t remember the cursor position for files larger than the huge file threshold.

# Tabs Preferences

On the Tabs tab in the Preferences screen, you can configure how tab pages work in EditPad Pro. Tabs enable you to quickly switch between files and projects. You can also quickly open and close files via the tabs.




## Tabs for Files and Projects

EditPad Pro can display up to two rows of tabs. One row for files, and another row for projects. The file tabs will only show the files in the active project. Switching between project tabs also switches between the files active in those projects.

Tabs are very convenient when working with a relatively small sets of files. When working with hundreds or even thousands of files, EditPad Pro's Files Panel is a more appropriate tool. If you prefer to use the Files

Panel, you can disable the tabs to save screen space. By default, EditPad always shows the tabs for files, but only shows the tabs for projects when you have more than one project open.

Pressing Ctrl+Tab on the keyboard switches between file tabs. If you turn on "use most recent order when switching tabs with Ctrl+Tab", then pressing Tab repeatedly while holding down Ctrl moves back through tabs you've previously activated, in reverse order. Unlike the Go|Back in Edited Files, Ctrl+Tab remembers all tabs you've activated, regardless of whether you edited those files.

When you release the Ctrl key after pressing Tab one or more times, the tab that is activated is moved to the last position in the list of recently activated tabs. So if you press Ctrl+Tab and release both keys and do this repeatedly, you'll be switching back and forth between two files. Essentially, Ctrl+Tab switches between files in EditPad like Alt+Tab switches between application in Windows.

If you turn off "use most recent order when switching tabs with Ctrl+Tab", pressing Ctrl+Tab moves through the file tabs from left to right, and pressing Shift+Ctrl+Tab moves through file tabs from right to left.

### Appearance

By default, EditPad will show images on tabs. File tabs show the icon associated with the file's extension. Project tabs show the Project|New Project icon for unmanaged projects, and the Project|Managed Project icon for managed projects.

The tab of the active file has a different color than the other tabs. If this distinction is not clear enough, you can either change the tab colors as explained below or turn on "active tab is taller than other tabs". When this option is selected, the active tab will have a larger height and a piece of background will be visible above all the other tabs. Turning off this option gives the tab control a much more compact look.

You can also change the font and font size used for the captions shown on the tabs. A bigger font is more readable but allows fewer tabs to fit on the screen.

### Location of the close button

EditPad can show an X button to make it easy to close tabs. You can have one close button at the right hand edge of the tab control, after the last tab. Clicking this closes the active tab. Alternatively, you can have a close button directly on the tab. You can have it on the active tab only, or on each tab. A close button on each tab makes it easy to close tabs with a left click, but you have to be careful not to accidentally close tabs when switching between them.

If you choose not to show the close button, you can still close the active tab with the File|Close command, or any tab by clicking on it with the mouse wheel. Not showing the close button allows more tabs to fit on the screen.

### Tab Colors

You can configure the colors of the tabs. Select an item in the list and click the Color and Text Color buttons to change the background and text colors of the selected tab style.

- Tab: Regular tab
- Active Tab: The tab of the file you are currently viewing
- Hot Tab: The tab underneath the mouse pointer
- Tab - modified: Tab of a modified file
- Active Tab - modified: The tab of the file you are currently editing
- Hot Tab - modified: The tab of a modified file underneath the mouse pointer
- Tab - read only: Tab of a read only file
- Active Tab - read only: The tab of the read only file you are currently viewing
- Hot Tab - read only: The tab of a read only file underneath the mouse pointer
- Tab button: Color of the X button directly on a tab
- Pressed tab button: Color of the X directly on a tab when you've clicked on it with the mouse
- Hot tab button: Color of the X button directly on a tab underneath the mouse pointer
- Tab control background: The empty space above and to the right of the tabs
- Tab control button: Color of the X and other buttons at the right end of the tab row
- Pressed tab control button: Color of the X or other button when you've clicked on it with the mouse
- Hot tab control button: Color of the X or other button underneath the mouse pointer
- Disabled tab control button: Color of disabled buttons at the right end of the tab row. The scrolling buttons will become disabled when you can't scroll further.

## Tab Actions

You can do more with the tabs in EditPad, beyond activating a file by clicking on its tab.

To quickly close a tab, move the mouse pointer over it and push down the wheel button on your mouse. If the file didn't have any unsaved changes, EditPad will close it instantly, without even activating it.

You can access other common file-related commands such as File|Save and File|Close All but Current quickly by clicking on one of the tabs with the right hand button of your mouse. If the tab you right-clicked on isn't the active tab, EditPad will activate the file first. If a command you often use isn't available in the right-click menu, you can add it by customizing the toolbars and menus.

Right-clicking a file tab also shows a few special commands such as the Read Only command for toggling the file's read-only status, and Copy Path to Clipboard for placing the file's full path onto the clipboard.

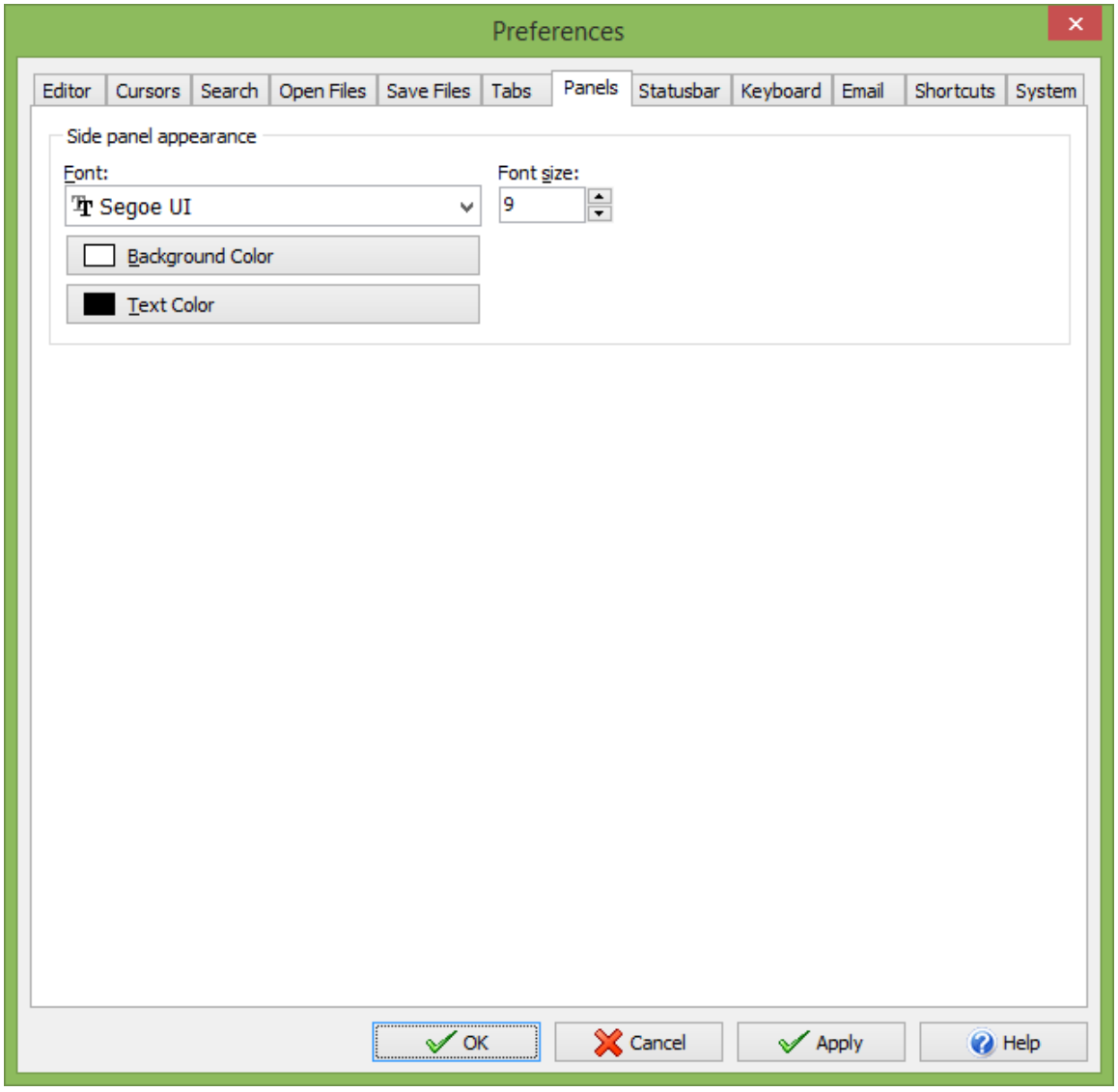
To quickly start with a blank, untitled file, double-click on the empty space after the last tab. Other common commands for opening files and commands affecting all files are available in the context menu that appears when you right-click on the empty space after the last tab. This menu is also configurable.

Right-clicking on a project tab shows a context menu with various commands to manage the project. You can configure this menu.

To quickly start with an empty project, double-click the empty space after the last project tab. This empty space too has a configurable context menu. It shows commands for opening projects and commands affecting all projects.

# Side Panel Preferences

On the Panels tab in the Preferences screen, you can change the appearance of EditPad's side panels.



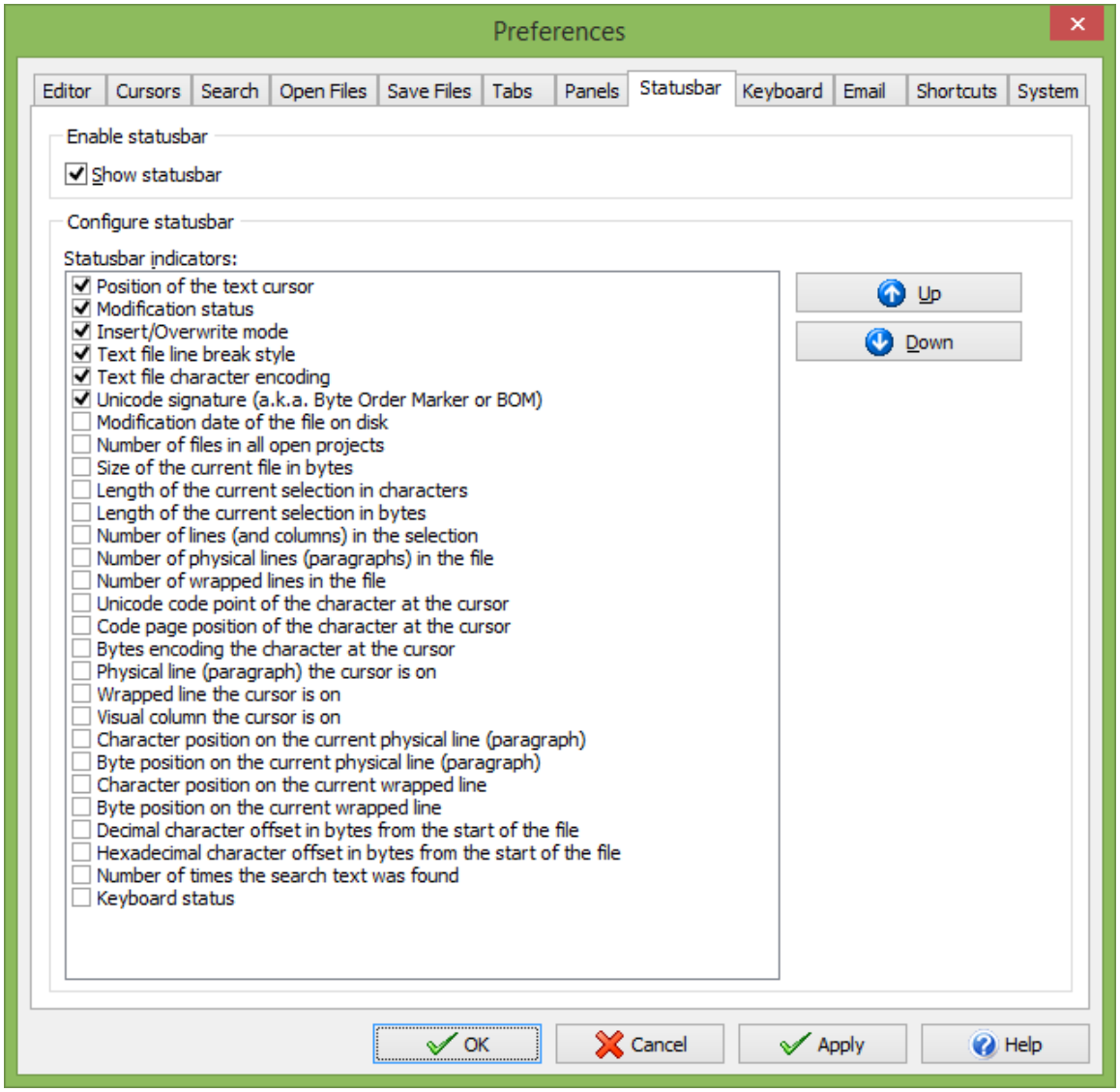

You can select a font face from the drop down list. Set the font size using the spinner box. Click the Background Color button to change the background of the side panels. Click the Text Color button to change the font's color.

These settings are used by the Clip Collection, Files Panel, Explorer Panel, FTP Panel, File History, and File Navigator panels.

These settings are not used by the Character Map and the Search Panel. Those panels use the same font and color as the file you're editing. The font and colors for files can be set via Options|Configure File Types.

# Statusbar Preferences

While both EditPad Lite and Pro have a status bar, it is only configurable in EditPad Pro. EditPad Pro can show many more indicators on the status bar. You may not want to clutter the screen with all of them. Select Options|Preferences in the menu, and click on the Statusbar tab.


Preferences
Editor
Cursors
Search
Open Files
Save Files
Tabs
Panels
Statusbar
Keyboard
Email
Shortcuts
System
Enable statusbar
Show statusbar
Configure statusbar
Statusbar indicators:
Position of the text cursor
Modification status
Insert/Overwrite mode
Text file line break style
Text file character encoding
Unicode signature (a.k.a. Byte Order Marker or BOM)
Modification date of the file on disk
Number of files in all open projects
Size of the current file in bytes
Length of the current selection in characters
Length of the current selection in bytes
Number of lines (and columns) in the selection
Number of physical lines (paragraphs) in the file
Number of wrapped lines in the file
Unicode code point of the character at the cursor
Code page position of the character at the cursor
Bytes encoding the character at the cursor
Physical line (paragraph) the cursor is on
Wrapped line the cursor is on
Visual column the cursor is on
Character position on the current physical line (paragraph)
Byte position on the current physical line (paragraph)
Character position on the current wrapped line
Byte position on the current wrapped line
Decimal character offset in bytes from the start of the file
Hexadecimal character offset in bytes from the start of the file
Number of times the search text was found
Keyboard status
Up
Down
OK
Cancel
Apply
Help

You can show or hide indicators on the status bar by marking or clearing the checkboxes at the left of the items in the list. You can also hide the status bar altogether by turning off "show status bar".

You can change the order in which the items are shown on the status bar by dragging and dropping them in the list. Click on an item, hold the mouse button down and drag it to the spot where you want it and release the mouse button. The topmost item in the list will become the leftmost item on the status bar.

The following status bar indicators are available:

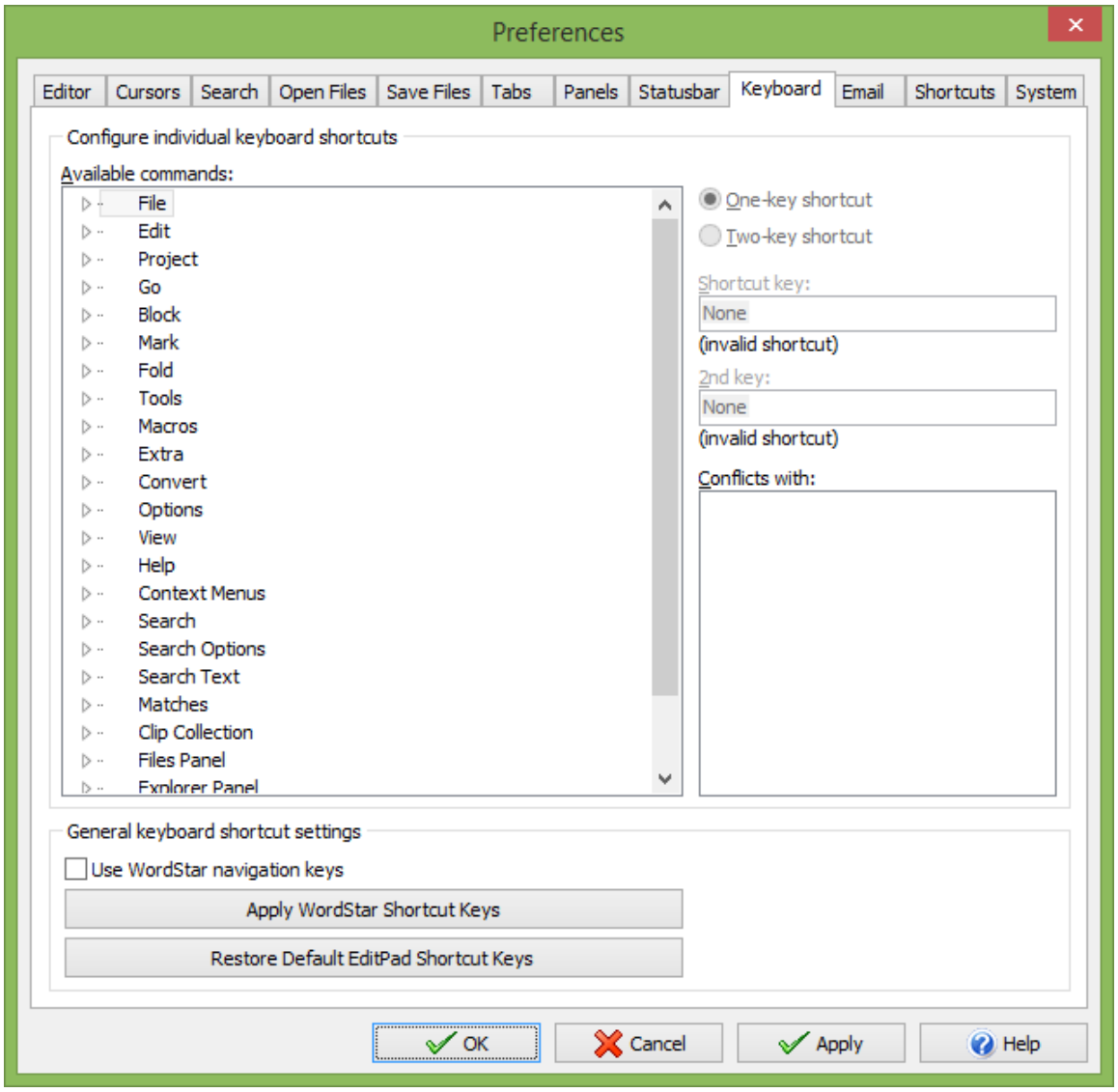
- Position of the text cursor: Number of the line and column the cursor is on. When word wrap is on, the line number depends on the "count physical lines only" setting in File Type|Editor Options. It is the same number shown in the margin when Options|Line Numbers is on. The column number indicates the visual column, which can differ from the number of characters to the left of the cursor. A single tab character may be multiple columns wide. Combining characters like diacritics may have no width. In hexadecimal mode, Clicking this indicator invokes Go|Go to Line or Offset.
- Modification status: Indicates "Modified" if the file has unsaved changes in EditPad. Indicates "Read Only" if the file is read-only. Otherwise the indicator is blank. Clicking it when blank makes the file read-only. Clicking it when showing "Read Only" makes the file editable.
- Insert/Overwrite mode: Indicates "Insert" when typing into the file pushes forward existing text to the right of the cursor. Indicates "Overwrite" when typing into the file overwrites existing text to the right of the cursor. Clicking this indicator toggles between insert and overwrite mode.
- Text file line break style: Indicates "Windows" if the file uses CRLF line breaks exclusively, "UNIX" if it uses LF line breaks only, and "Mac" if it uses CR line breaks only. If the file does not use any line break style exclusively, the indicator shows Windows, UNIX, or Mac to show the line break style that is used the most along with "(Mixed)" to indicate it is not used exclusively. Clicking this indicator shows a popup menu with the Convert|To Windows (CR LF), Convert|To UNIX (LF only), and Convert|To Macintosh (CR only) commands.
- Text file character encoding: Indicates the encoding EditPad uses to interpret the bytes in this file as characters. Clicking the indicator invokes Convert|Text Encoding.
- Unicode signature (a.k.a. Byte Order Marker or BOM): Indicates "BOM" if the file uses a Unicode encoding and has a Unicode signature or Byte Order Marker at the start of the file. Indicates "no BOM" if the file uses a Unicode encoding and does not have a Unicode signature. Indicates "---" if the file does not use a Unicode encoding.
- Modification date of the file on disk: The date and time that the file was last saved.
- Number of files in all open projects: Total number of files in all the projects that you have open in EditPad Pro. For managed projects, that includes open files, closed files, and outside files.
- Size of the current file in bytes: Total size in bytes the file would have on disk if you were to save it.
- Length of the current selection in bytes: Size of the selection in bytes. Depending on the encoding used by the file, the size in bytes may or may not correspond with the size in characters.
- Length of the current selection in characters: Number of characters in the selection. Tabs are counted as a single character. Combining characters such as diacritics are counted if they are typed separately from their base character.
- Number of lines (and columns) in the selection: Number of lines that are (partially) selected. When word wrap is on, all wrapped lines are counted, regardless of the "count physical lines only" setting in File Type|Editor Options. If the selection is rectangular, the indicator follows the number of lines with a colon and a second number indicating the number of visual columns in the selection. This is the width of the rectangular selection.
- Number of physical lines (paragraphs) in the file: Number of lines in the file, counted as if word wrapping were off.
- Number of wrapped lines in the file: When word wrap is on, this indicator counts the number of lines in the file, including wrapped lines. When word wrap is off, the number of physical lines is shown.
- Unicode code point of the character at the cursor: Indicates the Unicode code point of the character to the right of the cursor in hexadecimal and in decimal. This indicator works even when the file's encoding is not Unicode.
- Code page position of the character at the cursor: If the file uses a legacy code page, this indicator shows the index in that code page of the character to the right of the cursor in hexadecimal and decimal notation. If the file uses Unicode, the Unicode code point is indicated instead.
- Bytes encoding the character at the cursor: Indicates the actual bytes in the file that represent the character to the right of the cursor. The bytes are shown in hexadecimal notation like you would see

in hexadecimal mode. For encodings that store files as pure ASCII using character escapes for non-ASCII characters, this indicator shows the ASCII text used to escape non-ASCII characters.

- Physical line (paragraph) the cursor is on: Number of the line the cursor is on, counting lines as if word wrap were off.
- Wrapped line the cursor is on: Number of the line the cursor is on. When word wrap is on, wrapped lines are counted too, regardless of the "count physical lines only" setting in File Type|Editor Options.
- Visual column the cursor is on: The visual column position of the cursor. This can differ from the number of characters to the left of the cursor. A single tab character may be multiple columns wide. Combining characters like diacritics may have no width.
- Character position on the current physical line (paragraph): Number of characters on the line to the left of the cursor. When word wrap is on, all characters before the cursor up until the previous line break in the file are counted, even if those characters are wrapped into multiple lines. Tabs are counted as a single character. Combining characters such as diacritics are counted if they are typed separately from their base character.
- Character position on the current wrapped line: Number of characters on the line to the left of the cursor. When word wrap is on, only the characters before the cursor on the same wrapped line are counted. Tabs are counted as a single character. Combining characters such as diacritics are counted if they are typed separately from their base character.
- Byte position on the current physical line (paragraph): Number of bytes on the line to the left of the cursor. When word wrap is on, all characters before the cursor up until the previous line break in the file are counted, even if those characters are wrapped into multiple lines.
- Byte position on the current wrapped line: Number of bytes on the line to the left of the cursor. When word wrap is on, only the characters before the cursor on the same wrapped line are counted.
- Decimal character offset in bytes from the start of the file: Decimal representation of the number of bytes in the file before the cursor position, counting all bytes from the start of the file.
- Hexadecimal character offset in bytes from the start of the file: Hexadecimal representation of the number of bytes in the file before the cursor position, counting all bytes from the start of the file.
- Number of times the search text was found: Incremented by commands such as Search|Find Next and Search|Find Previous. Reset to zero (or one if a match is found) by commands like Search|Find First. Shows the total number of matches found by commands like Search|Replace All and Search|Copy Matches.
- Keyboard status: If you have pressed the first key in a two-stage keyboard shortcut then the first key is shown in this status bar indicator. If not, the indicator shows NUM, CAPS, and/or SCROLL if the Num Lock, Caps Lock, and/or Scroll Lock keys on the keyboard are active.

# Keyboard Preferences

On the Keyboard tab in the Preferences screen, you can change the shortcut key combination for each item in EditPad's main menu.




## Configure Individual Keyboard Shortcuts

To assign a new shortcut key combination to a menu item, select the menu item from the list. Then click on the box labeled "shortcut key". Press the key combination you want on the keyboard. To remove a key combination from an item, click on the "shortcut key" box and press the delete or backspace key on the keyboard.

Each shortcut key combination can be used by only one menu item. If the shortcut key combination you assigned to the currently selected menu item is already used for one or more other menu items, those other menu items will be listed in the box labeled "conflicts with".

You can also assign shortcuts keys to tools and to macros when configuring tools and recording macros. You cannot assign the same shortcut key to both a menu item and a tool or macro.

If you run out of key combinations, EditPad also supports two-key keyboard shortcuts. When you press the first key combination of a two-key shortcut, EditPad will do nothing except remember that you pressed that key. The next key you press is then taken as the second key. If it forms a valid two-key shortcut, the corresponding action is executed. If not, neither key press will have any effect.

To configure a two-key shortcut, click on the "two-key shortcut" radio button. Then click on the "shortcut key" box and press the first key combination. The first key must either be a Ctrl+Letter combination (possibly including Shift and/or Alt), or a function key (possibly including Ctrl, Shift and/or Alt). Up to 15 different key combinations can be used as the first key in a two-key pair. Then click on the "2nd key" box and press the second key combination in the two-key shortcut. The second key combination can be any key, including a letter without a modifier (Ctrl, Shift or Alt). If you specify a letter without a modifier, both the letter without the modifier and the letter with the same modifiers as the first key will work. E.g. if you assign Ctrl+K, B to the Begin Selection command, you can begin a selection by pressing Ctrl+K followed by B, or Ctrl+K followed by Ctrl+B to begin a selection. Specifying a letter without modifiers as the second key makes it easier to quickly press both keys, since it doesn't matter if you release the Ctrl key before the second key (the letter B in the example) or after it.

When using two-key keyboard shortcuts, it is a good idea to add the keyboard status indicator to the status bar. When you press the first key in a two-key combination, the keyboard indicator will show that key. That way you know the next key you press will be interpreted as the second key in the combination. After you press the second key, the indicator reverts to showing num lock, caps lock and scroll lock.

### General Keyboard Shortcut Settings

EditPad supports the classic Wordstar navigation keys for people who are used to them. WordStar is an old word processor that used Ctrl+Letter key combinations to navigate through the document. WordStar was created at a time when most keyboards did not yet have separate arrow key blocks. The checkbox only enables the navigation keys. It does not change any of the key combinations that you assigned to EditPad's menu items. Note that many of these navigation keys conflict with standard Windows shortcut keys. E.g. Ctrl+S is the standard Windows shortcut for File|Save. In WordStar, Ctrl+S moves the text cursor one position to the left, just like the left arrow key.

Click the "Apply WordStar Shortcut Keys" button to assign WordStar key combinations to various menu items. E.g. Ctrl+K, S will be assigned to File|Save. Shortcut keys assigned to EditPad menu items that don't have a WordStar equivalent will not be changed. The assignment is a one-time event. You can freely reconfigure any of the changed keys.

Click the "Restore All Default Shortcut Keys" button to restore the default EditPad shortcuts for all menu items. It will also turn off the option to use WordStar navigation keys, since many of those conflict with EditPad's default shortcuts.

The keyboard navigation keys cannot be configured. They are always available, even when you've enabled WordStar navigation keys.

# Wordstar Navigation Keys

EditPad Pro supports the classic Wordstar navigation keys for people who are used to them. WordStar is an old word processor that used Ctrl+Letter key combinations to navigate through the document. WordStar was created at a time when most keyboards did not yet have separate arrow key blocks. Note that many of these navigation keys conflict with standard Windows shortcut keys. E.g. Ctrl+S is the standard Windows shortcut for File|Save. In WordStar, Ctrl+S moves the text cursor one position to the left, just like the left arrow key.

You can enable the WordStar navigation keys in the Keyboard Preferences. When enabled, they are recognized by every full text editor control in EditPad, such as the main editor, the search box, the replace box, the clip editor, etc.

The shortcuts with Ctrl+Q are two-key combinations. First, press Ctrl+Q. Then release the Q key and press the second letter in the key combination. Whether you press Ctrl along with the second key or not makes no difference.

### Cursor movement keys

Ctrl+S moves the text cursor one position to the left.
Ctrl+D moves the text cursor one position to the right.
Ctrl+E moves the text cursor one line upward.
Ctrl+X moves the text cursor one line downward.
Ctrl+A moves the text cursor to the start of the previous word or the end of the previous line, whichever is closer.
Ctrl+F moves the text cursor to the start of the next line or the end of the current line, whichever is closer.
Ctrl+R moves the text cursor up an entire screen.
Ctrl+C moves the text cursor down an entire screen.

Ctrl+W scrolls down one line. Cursor moves up one line unless it is already at the top (configurable).
Ctrl+Z scrolls up one line. Cursor moves down one line unless it is already at the top (configurable).

Ctrl+Q, S moves the text cursor to the beginning of the line (configurable).
Ctrl+Q, D moves the text cursor to the end of the line.

Ctrl+Q, E moves the text cursor to the top of the screen.
Ctrl+Q, X moves the text cursor to the bottom of the screen.
Ctrl+Q, R moves the text cursor to the start of the file.
Ctrl+Q, C moves the text cursor to the end of the file.

### Editing commands

Ctrl+I inserts a tab character.
Ctrl+N inserts a line break.

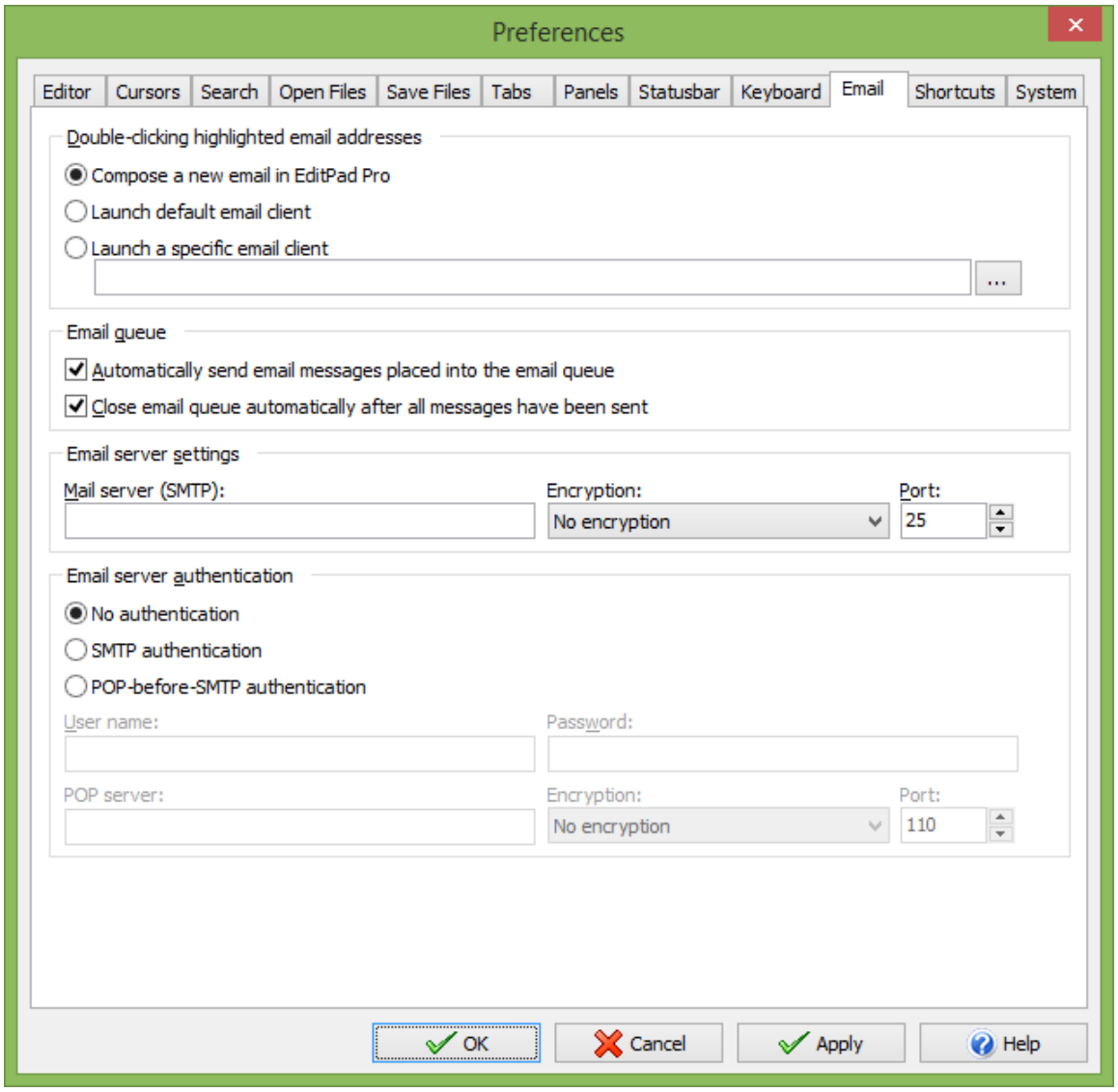
Ctrl+G deletes the character to the right of the text cursor.
Ctrl+H deletes the character to the left of the text cursor.
Ctrl+T deletes the part of the current word to the right of the text cursor. If the cursor is not on a word, all characters to the right of the cursor up to the start of the next word are deleted.

Ctrl+Q, Y deletes all the text on the current line to the right of the text cursor.
Ctrl+Q, T deletes all the text on the current line to the left of the text cursor.

Ctrl+V toggles between insert and overwrite mode.

# Email Preferences

Select Options|Preferences in the menu and click on the Email tab to configure EditPad Pro for sending email.




## Double-Clicking Highlighted Email Addresses

When you double-click a highlighted email address in a text file, EditPad Pro can do one of three things:

- Compose a new email in EditPad Pro. This is the same as selecting File|Mail in the menu and entering the email address.
- Launch default email client. EditPad Pro will attempt to launch the application associated with the mailto: URL protocol on your computer.

- Launch a specific email client. Click on the (...) button to select the .exe file of the email software you want EditPad Pro to launch. You can also type in the command line directly and specify command line parameters. If you enter %EMAIL% on the command line, EditPad Pro will substitute that placeholder with the actual email address. If you don't enter %EMAIL% on the command line, EditPad Pro will launch the .exe with the email address as the sole command line parameter.

## Email Queue

If you turn on "automatically send email messages placed into the email queue", EditPad Pro will start sending email right away when you click the Send button in the email composition panel. This way you do not have to click on the Send button in the mail queue. You should turn on this option if you have a permanent Internet connection.

Turn on "close email queue automatically after all messages have been sent" to make the mail queue disappear automatically when all emails have been sent successfully. If there was a problem sending some or all of the messages in the queue, then the queue will remain visible, so you can see which messages were not sent.

## Email Server Settings

Before you can use File|Mail, you need to specify the mail server EditPad Pro should use to send email. EditPad Pro only supports the SMTP protocol, which is the de facto standard for sending email on the Internet. If you connect to the internet through a public dial-up or broadband ISP, you can use your ISP's SMTP server. Typical names for the SMTP server are mail.isp.com, relay.isp.com and smtp.isp.com, where isp.com is your internet provider's domain name. The standard port for SMTP connections is 25.

Most SMTP servers do not use encryption. Some require SSL or TLS encryption, and some allow the email client to negotiate for TLS encryption. Your email provider should tell you which encryption method you need to select. If they don't, you may be able to guess the encryption method from the port number your server uses. Port 465 is the standard port for SMTP connections encrypted with SSL. Port 587 is the standard port for SMTP connections encrypted with TLS. If you select one of these encryption methods in EditPad, the port number is changed automatically. You can still choose a different port number. If your SMTP server uses encryption on port 25, set the encryption method to "TLS, if available".

For Gmail, set the server to `smtp.gmail.com` and use TLS encryption on port 587.

If you connect to the Internet through a corporate network, you will have to ask your network administrator which server you can use. MS Exchange and other proprietary protocols are not supported by EditPad Pro.

## Email Server Authentication

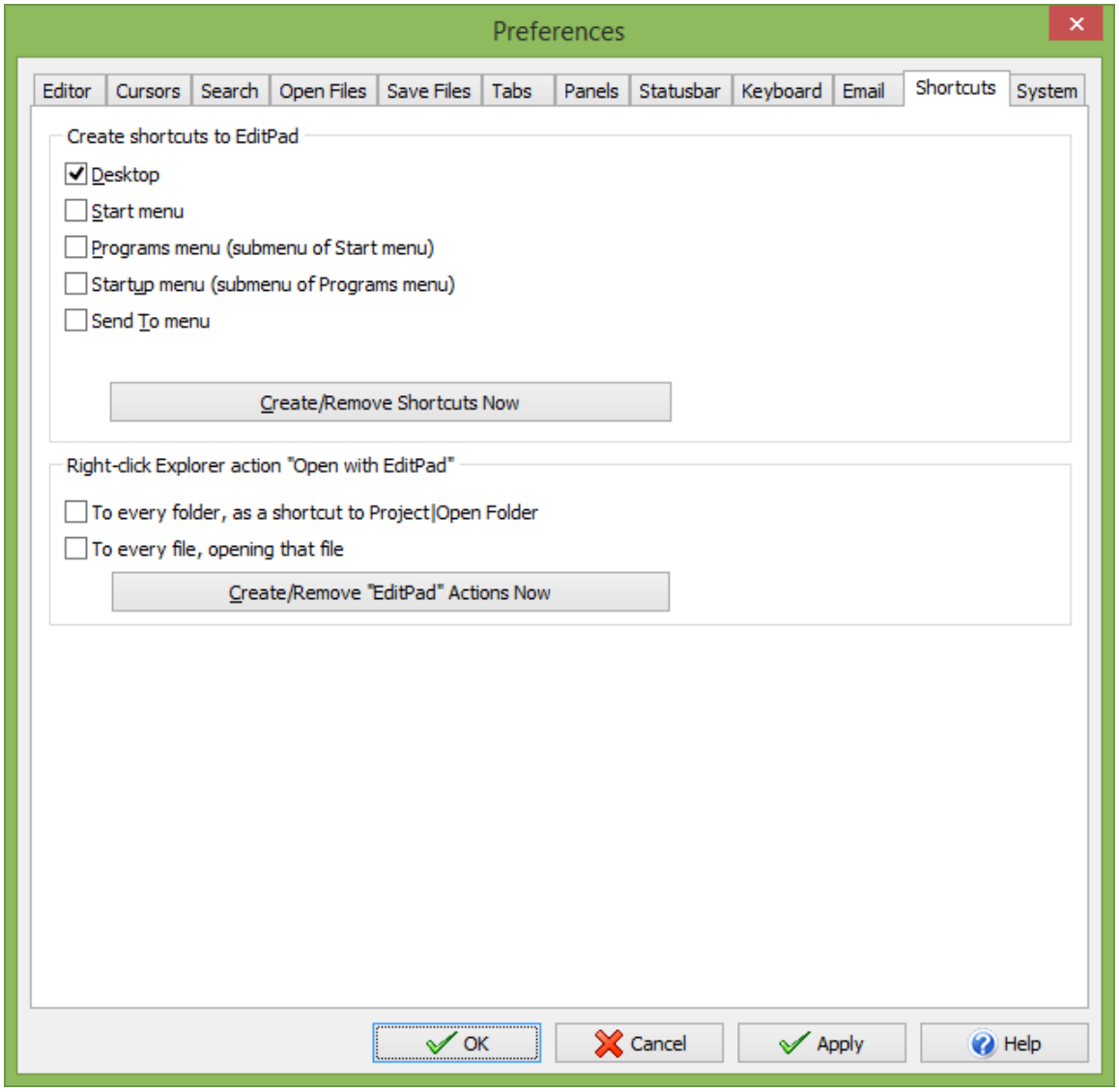
Many SMTP servers only allow you to send email after logging in with your email username and email password. Often, the requirement is that you check incoming your email before you can send email. This is called "POP-before-SMTP authentication". When you mark this option, you can type in your email username and email password, as well as the server from which you retrieve incoming mail. EditPad Pro supports the POP3 protocol, which is the de facto standard for receiving email on the Internet. Typical server names are mail.isp.com, pop.isp.com and pop3.isp.com. The standard port is 110 for unencrypted POP3 and TLS-encrypted POP3. The standard port is 995 for SSL-encrypted POP3. Note that EditPad Pro will not actually

download any of your email. It will only log onto the server, and then disconnect, for the purpose of authenticating you so you can send email using EditPad Pro.

If your ISP requires a username and password to send email, but do not require you to check your incoming email before sending email, the ISP is using "SMTP authentication". When you mark this option, you can type in your email username and email password that EditPad Pro can use to connect to the SMTP server.

For Gmail, select "SMTP authentication" and enter your full `@gmail.com` email address and your Gmail password.

# Shortcuts Preferences




### Create Shortcuts to EditPad

This page in the Preferences window makes it easy for you to create shortcut icons to EditPad. Select the various places where you want a shortcut icon and click the "Create/Remove Shortcuts Now" button. To remove a shortcut, clear the appropriate checkboxes and press the same button.

If you place a shortcut in the "Startup menu (submenu of Programs menu)", the green EditPad icon will automatically appear next to the system clock whenever you turn on your computer.

Put a shortcut in the “Send To menu” so you can quickly open any file in EditPad from within Windows Explorer by right-clicking on the file, selecting Sent To from the context menu and then selecting EditPad.

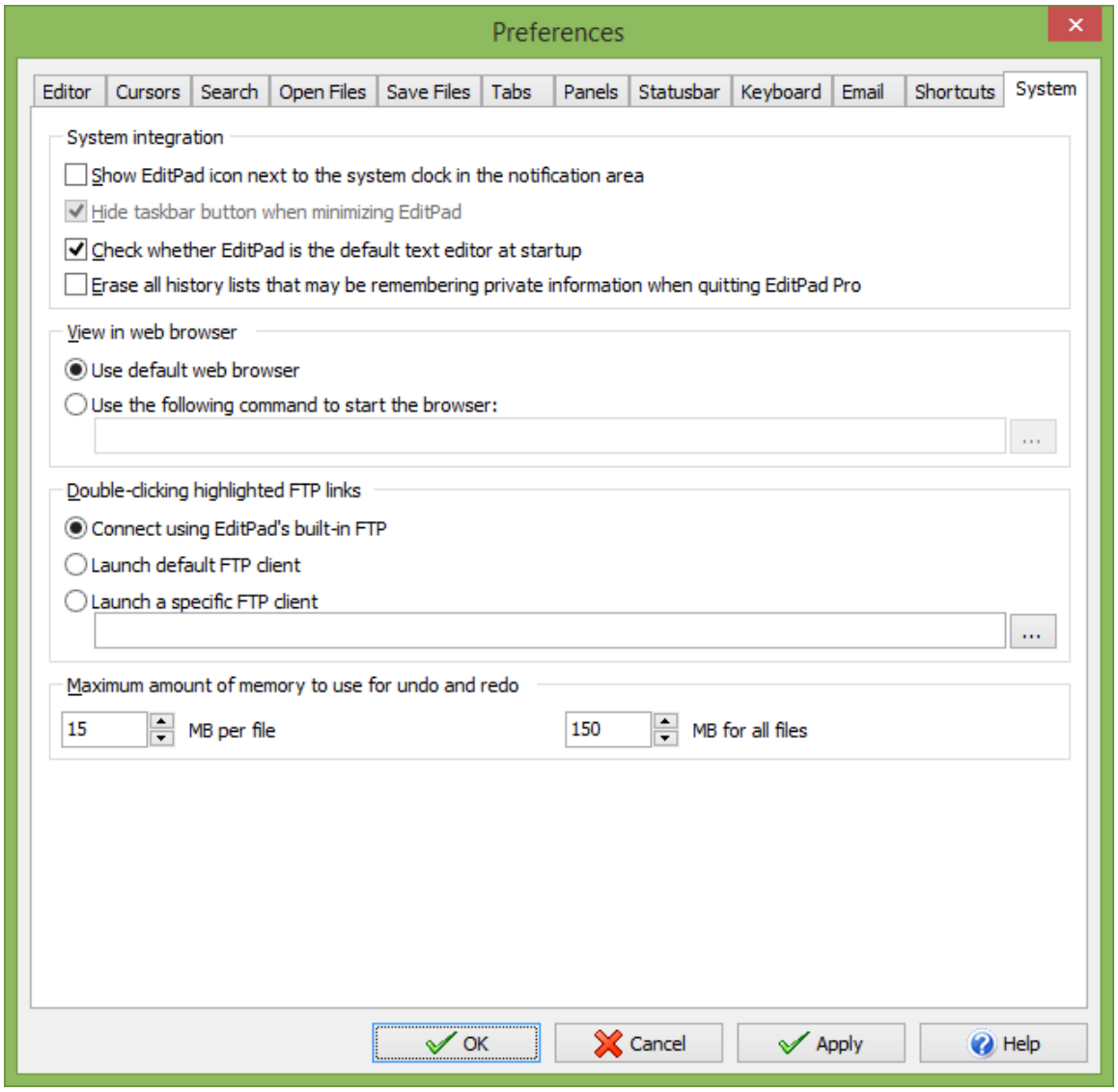
The “Quick Launch Toolbar” is not available in Windows 7.

### Right-Click Explorer Action

EditPad Pro can also add itself to the context menu of every file and/or every folder in Windows Explorer. The context menu is the menu that appears when you click on an item with the right hand mouse button. Select the context menus in which you want EditPad Pro to appear, and click "Create/Remove EditPad Actions Now".

# System Preferences

On the System page in Options|Preferences, you can make some choices how EditPad Pro interacts with the rest of your computer and network.




## System Integration

If "Show EditPad icon next to the system clock in the notification area" is activated, a green EditPad icon will be visible next to the system clock (this area is called the "notification area") when EditPad is running. When you close EditPad in this situation, it will not really close but only hide itself so the system tray icon remains visible. You can then make EditPad quickly reappear by clicking on the system tray icon. You can right-click on it to quickly start with a new file or open an existing one. To completely shut down EditPad, select File|Exit in the menu.

Turn off this option to make EditPad behave like a regular Windows program.

When the "Show EditPad icon next to the system clock in the notification area" option is activated, you can choose if you want EditPad's button on the taskbar to hide or not when you minimize EditPad by marking or clearing "Hide taskbar button when minimizing EditPad". Note that the taskbar button will always hide when you close EditPad.

Turn on "check if Editpad is the default text editor at startup" if you want EditPad to check whether it is still associated with `.txt` files each time you start it. If this option is on, and EditPad detects it is no longer associated with `.txt` files, it will ask you whether you want EditPad to associate itself with `.txt` files or not.

Turn on "erase all history lists that may be remembering private information when quitting EditPad Pro" to make Editpad Pro forget the lists of recent items kept by these commands:

> File|Open File|Mail Project|Open Project Project|Open Folder Project|Import File Listing Search|History Search|Find on Disk Explorer Panel|Set Home Folder Explorer Panel|Filter FTP Panel|Connect FTP Panel|Filter

### View in Web Browser

If the View|Browser doesn't properly detect your web browser, you can specify a specific application to be used as the web browser. Click the (...) button to browse for the .exe file of the application you want to use. You can also manually enter a command line. If you include %URL% as part of the command line parameters, that placeholder will be substituted with an http:// or file:// URL to the file to be opened. If you don't enter %URL% on the command line, EditPad Pro will launch the .exe with the URL as the sole command line parameter.

### Double-Clicking Highlighted FTP Links

EditPad Lite will always open FTP links in your computer's default FTP client. EditPad Pro provides three choices. The default is to use EditPad's built-in FTP. This is the most comfortable option to edit files stored on FTP servers. The "launch default FTP client" runs whichever application is configured on your computer to handle ftp: URLs.

Finally, you can specify a specific application to be used as the FTP client. Click the (...) button to browse for the .exe file of the application you want to use. You can also manually enter a command line. If you include %URL% as part of the command line parameters, that placeholder will be substituted with an ftp:// URL to the file to be opened. If you don't enter %URL% on the command line, EditPad Pro will launch the .exe with the URL as the sole command line parameter.

### Maximum Amount of Memory to Use for Undo and Redo

As explained in the help topic for the Edit|Undo command, EditPad normally keeps an undo history for all changes you've made to all files that you have open in EditPad since you opened them. During very heavy editing sessions, this may cause the undo history to use too much memory. As a safeguard, EditPad will automatically discard the oldest changes from the undo history when it grows too large. In EditPad Lite these

limits are automatically set based on the amount of RAM your PC has. In EditPad Pro, you can configure the limits yourself.

The limit per file limits the undo history for each individual file. If the undo history for a file grows beyond this point, the oldest changes for that file are discarded from its undo history. EditPad does this even if there are older changes in other files, and even when the total undo history size for all files has not yet exceeded the limit for all files.

The limit for all files limits the combined size of the undo history for all open files. If the total undo history size grows beyond this point, the oldest changes are discarded from the undo history. EditPad does this regardless of which file(s) the oldest changes were made in, and regardless of how much memory the undo history for those files uses.

Set the limit for all files to allow enough free memory for the files themselves that you edit in EditPad, as well as for Windows and for all the other applications you're running. Set the limit per file lower than the limit for all files to make sure that making lots of changes to one file does not clear out the undo histories for all the other files you have open in EditPad.

# 16. View Menu

## View|Clip Collection

Select Clip Collection from the View menu to open EditPad Pro's Clip Collection pane. There you can store common blocks of text and templates for quick repeated reuse. When editing complex documents, move chunks of text into the collection for temporary storage. The Clip Collection is also a great place to jot down notes and ideas, and to keep important information at your fingertips.

AceText, a product available separately, is specifically designed to enable you to very flexibly work with large numbers of text snippets. If you have installed AceText, EditPad Pro will use AceText's Clip Collection functionality, and display the collection that is active in AceText. EditPad Pro and AceText will automatically keep the collection synchronized when you edit it in AceText or EditPad Pro.

If you do not have AceText, EditPad Pro will manage the collection of clips by itself. You may want to check out the free evaluation version of AceText. AceText offers a lot of extra functionality. Most importantly, AceText enables you to use your clip collections in combination with applications other than EditPad Pro. AceText cooperates with almost all Windows software.

### Adding Text to The Collection

To add text to the collection, simply select it in the editor and click the "New Clip" button in the toolbar above the Clip Collection. You can also drag and drop the selection from the editor into the collection. Drag and drop clips inside the collection to rearrange them.

You can edit a clip's label or view and edit the text it holds by clicking the Edit Clip button. If you use AceText, this will activate the AceText Editor. If not, you can edit the clip directly in EditPad Pro.

If you want to add many clips to a collection, organize them in folder to make it easier to find the right clip later. Simply click the New Folder button to create a new folder. Add clips to the folder by selecting the folder before clicking the New Clip button. You can also drag and drop text or clips onto the folder.

### Using Clips from The Collection

To insert the contents of a clip into the file you are editing, simply double-click it or drag-and-drop it onto the spot where you want to insert it. If the clip holds before and after text, it will be placed around the text you selected in the editor. If you did not select any text, the before and after texts are inserted right next to each other, and the text cursor is placed between them. Before and after text makes it very easy to insert common text blocks that have an opening and closing part, such as HTML or XML tags.

### Opening and Saving Collections

If you are not using AceText, EditPad Pro will automatically save Clip Collections. This way, you never have to worry about losing any clips. The Save button allows you to save the collection under a different name. To

open a collection later, click the Open button and select the file, or click the downward pointing arrow next to the Open button to select a collection you recently worked with.

To make Clip Collections even more transparent, you can assign a default collection to each file type in the file type configuration. That collection is then automatically opened whenever you switch to a file of that type. E.g. you can assign a collection with HTML tags to the HTML file type. That way, the clip collection will always be ready for inserting HTML tags whenever you edit an HTML file.

EditPad Pro can also import TextPad Collection Library files (*.tcl files). After clicking the Open button on the Clip Collection pane's toolbar, select the "TextPad Collection Libraries" file type in the drop-down list at the bottom of the file selection screen. EditPad Pro cannot save TextPad Collection Library files.

### Sharing Collections

Click the Share Collections button to download Clip Collections shared by others, or to upload your own collections. A new window will pop up. EditPad Pro will connect to the Internet immediately to download the list of available collections. If your computer can only access the internet via a proxy server, click the Settings button in the lower left corner of the screen to enter your proxy server's settings.

The Download tab shows all available collections, including the ones you uploaded. The list shows the collection's title, the file name used by the author, how many clips it contains, who shared it, when it was first shared and when it was last updated. Click on a collection to get a more detailed description, and the author's name and email address (if provided by the author). Click the Download button to download the selected scheme.

The Share tab shows the collections that you previously uploaded. EditPad Pro automatically uses your EditPad Pro license to identify you. You can check your license details in the about box. To share a collection, select its file, and enter a title and your name. You can also enter your email address and web site URL if you want. In the Description box, explain what your collection is about. Click the Upload button to upload it. If you select a collection you previously uploaded, EditPad Pro will fill out all the fields with what you entered when you uploaded that collection. This way you can easily update a previously shared collection, or use the same details for another one. If you upload a collection with the same file name as a collection you already shared, the newly uploaded collection will replace the old one. If you upload a collection with another file name, it will be added as a new collection, even if you give it the same title or description as another one. Click the Delete button to stop sharing a collection.

EditPad Pro downloads and uploads in the background. The Progress tab shows all collections in the queue, and recently completed transfers. If an error occurs, EditPad Pro will not pop up a message box. The Progress tab will indicate the error.

## Edit Clip

To edit a clip in the Clip Collection, select the clip, and click the Edit Clip button in the Clip Collection toolbar.

A clip consists of the following parts:

**Label:** The label that identifies the clip in the collection. If you do not type in a label, the start of the text itself is used as the label.

**Kind:** EditPad Pro supports different kinds of clips. "Plain text" clips hold ordinary text. A word, a sentence, several paragraphs, or a complete text document of any size. "Rectangular text block" clips hold a rectangular selection. "Binary data" clips hold data in hexadecimal format. If you create a clip while EditPad Pro is in hexadecimal mode, the clip will hold "binary data".

"Before and after text" are a special, very useful kind of clip. You cannot create such a clip by dragging and dropping some text onto the Clip Collection. You can only create them by editing a clip, or by clicking the New Clip button when no text is selected. These clips are very useful to insert frequently used bits of text that consist of an opening and a closing part, such as HTML tags. The before part is placed before the cursor or selection, and the after part is placed behind it.

**Indent:** Turn on the indent option for "plain text" and "before and after text" clips, if the clip consists of multiple lines, and you want the second and subsequent lines to be lined up with the first line. E.g. source code is often indented to indicate block structure. Turn on the "indent" option for clips holding source code snippets, and they will be properly indented when you use them.

**Text:** The text stored by the clip. You can use all text editing keyboard shortcuts and mouse actions to edit the text of a clip.

## View|Character Map

Select Character Map in the view menu to show a grid with all the characters available in a particular text encoding. Read the help topic for Convert|Text Encoding to learn more about text encodings. By default, the character map shows the characters supported by the encoding used by the active file. You can select a different encoding in the drop-down list on the character map's toolbar. Selecting an encoding in the character map only affects the character map. It does not affect the file you're editing.

If the selected encoding is an 8-bit encoding, the character map displays all characters in the encoding except non-printable control characters. The Windows, Mac, and DOS encodings, except those for Far East languages, are encodings with 240 characters and 32 control characters. The ISO-8859 and EBCDIC encodings are encodings with 192 characters and 64 control characters. Some character maps have holes in them indicated by crossed-out grid cells. These indicate positions in the encoding that do not define any character.

The various Unicode transformations define thousands of characters. So do the Windows, Mac, and EUC encodings for Korean, Japanese and Chinese. EditPad's character map displays the characters in the order given to them by the Unicode standard, even for the Windows, Mac, and EUC encodings. To make it easier for you to find the character you want, you can filter the map to show only characters of a certain type. You can filter by Unicode category, Unicode block, and/or Unicode script. To enable a filter, simply select a choice from the drop-down list. To disable it, select (all categories), (all blocks), or (all scripts) at the top of the list. If you enable multiple filters, all of them apply at the same time. If you select the "decimal digits" category and the Thai block, the map will show Thai digits only (or nothing at all if the encoding doesn't support Thai).

If certain characters appear as squares, that means the character cannot be displayed using the current font. You can select a different font with Options|Font. Particularly when showing all Unicode characters, you

may see a lot of squares. While Unicode tries to define characters for all human languages and scripts, most fonts only support one script, or one group of closely related scripts. Microsoft supplies a font called "Arial Unicode" with recent versions of Windows. This font can display nearly all Unicode characters.

When you hover the mouse over a character in the map, its decimal and hexadecimal number in the encoding will be shown in the status bar. Double-click on a character to insert it into the file.

Because you can make the character map show any encoding, it is possible that the character map shows characters that cannot be represented by the encoding used by the active file. If you try to insert an unsupported character, it will show up as a question mark in the file, and EditPad will show a warning that the character could not be inserted. The question mark is not a placeholder but a permanent question mark. Press Backspace or use Edit|Undo to remove the question mark. To insert the proper character, change the file's encoding first, and then insert the character again.

To change the file's encoding, click the button next to the drop-down list with the encodings. A menu will pop up. The Text Encoding item at the bottom is identical to the Text Encoding item in the Convert menu. It pops up a window that allows you to change the encoding. If you have already selected the encoding you want in the drop-down list on the character map, you can use the Display File with Encoding item or the Convert File to Encoding item to change the file's encoding without using the Convert|Text Encoding popup window. The Display File with Encoding item corresponds with the "interpret the data as being encoded with another character set" choice in the Text Encoding window. The Convert File to Encoding item corresponds with the "encode the original data with another character set" choice. Both choices are explained in detail in the Convert|Text Encoding help topic.

The left hand side of the character map toolbar has several buttons to insert various representations of the character you selected in the character map. The leftmost button inserts the character itself. The next two buttons insert the character's code page index in decimal and hexadecimal notation. These buttons are only available when the character map shows an 8-bit code page. They insert the character's position in the code page being shown by the character map. This is not necessarily the character's index in the code page being used by the file you're editing.

The remaining 3 buttons insert the Unicode code point of the selected character in 3 different notations: a Unicode escape, a decimal numeric character reference, or a hexadecimal numeric character reference.. They always insert the Unicode code point, regardless of the encoding used by the file or the encoding shown in the character map. For example, if you have the euro symbol selected in the character map, these buttons insert `\u20AC`, `€`, and `€` because the euro symbol occupies code point U+20AC in the Unicode standard.

The edit box at the right hand side of the character map toolbar allows you to look up characters in the character map. To look up a character, type in the character itself or its representation, and press Enter. The "look up" box supports various notations. If you type in a single character, that character is selected in the character map. If you type in a Unicode escape such as `\u20AC` or a numeric character reference such as `€` or `€` then the character represented by that Unicode code point is selected in the character map. This works even when the character map is showing a non-Unicode encoding. You can also type in just a decimal number such as `169` or just a hexadecimal number such as `A9`. If you want `80` to be taken as a hexadecimal number, type in `0x80` or `80h`. How the number is interpreted depends on whether the character map is displaying an 8-bit encoding or not. If the encoding is 8-bit, the drop-down lists with Unicode categories, blocks, and scripts will be invisible. Then the number is interpreted as an index to that 8-bit code page. In the Windows code pages, for example, `0x80` is the euro symbol. If the character map encoding is not 8-bit, meaning the 3 Unicode drop-down lists are visible, then the number is taken as a Unicode code point, even if the encoding is not Unicode. If you have the multi-byte Windows 932 code page selected, `20AC` is

interpreted as the euro symbol, while `0x80` is interpreted as the control character represented by code point U+0080, even though the euro symbol is represented by the single byte `0x80` in code page 932.

## View|Hexadecimal

Click on the Hexadecimal item in the View menu or press Ctrl+H to switch between text and hexadecimal editing. The undo and redo lists are be cleared whenever you switch.

In hexadecimal mode, the editing area is split into three zones. The left zone is a column with byte offsets. It indicates the offset relative to the start of the file of the first byte on each row in the hex editor. The first file in the byte has offset zero.

The middle area shows all the bytes in the file represented as a hexadecimal number. You can type in the numbers 0 through 9 and the letters A through F to insert bytes. To change some bytes without changing the length of the file, press the Insert key on the keyboard to toggle between Insert and Overwrite mode. In Overwrite mode, the bytes you type in will replace the bytes already present in the file.

The right hand column shows the textual representation of each byte. You can select the code page used to translate the bytes into characters in the Editor Preferences. You can type characters into the text column. EditPad Pro will use the selected code page to translate them into bytes.

If you want to see only hexadecimal representation or only the textual representation, select Hexadecimal Only or ASCII only in the submenu of the Hexadecimal item in the View menu. Select Hexadecimal and ASCII to restore the default. Whichever mode you last selected in the submenu is used as the default next time you click the View|Hexadecimal item directly.

If you have used View|Split Editor to split the editor in two, then you can select Split Hexadecimal and ASCII in the View|Hexadecimal submenu to have the first half of the editor show only the hexadecimal representation, and the second half of the editor only the ASCII representation. If you didn't split the editor, then the editor shows both the hexadecimal and ASCII representations.

By default, the hexadecimal view groups bytes in blocks of 8 bytes and shows as many blocks of 8 bytes on one row as will fit within the width of EditPad's window. If your binary files use a specific record size, it may be more convenient to display one record on one row, even if that requires horizontal scrolling or leaves blank space. Use Options|Record Size to specify the number of bytes EditPad Pro should display per row in hexadecimal mode.

If your binary files store integer or floating point numbers that are larger than one byte, you can easily edit those numbers in decimal representation using EditPad Pro's Byte Value Editor while you're in hexadecimal mode.

## View|Byte Value Editor

When editing a file in hexadecimal mode, you can open EditPad Pro's Byte Value Editor. With the Byte Value Editor, you can edit the byte that the text cursor points to in the hexadecimal editor, and up to 7 bytes that follow it. Instead of entering hexadecimal values directly in the hex editor, you can enter decimal numbers in various formats. When you enter a number in one of the formats, EditPad Pro automatically

converts the decimal number into 1 to 8 bytes. EditPad Pro uses those bytes to replace the byte pointed to by the cursor and as many following bytes as needed.

The Byte Value Editor works with the main file editing area, as well as with the search and replace boxes. The edit box that was the last one to have keyboard focus is the one the Byte Value Editor is connected to.

The first row of values are unsigned integers of 1, 2, 3, 4, 6 or 8 bytes. Values are directly converted between decimal and hexadecimal.

The second row of values are signed integers of the same sizes. Hexadecimal values are negated using two's complement, like modern microprocessors do. E.g. if you enter -1 in any of the signed integer boxes, all bytes will be set to FF.

The binary box allows you to enter up to 8 bits to edit the current byte as a binary number. This is useful if the byte represents a bit field.

Three floating point boxes allow you to enter floating point numbers of various precisions. The 4-byte floating point number defines a 32-bit single precision floating point number following the IEEE standard. The 8-byte floating point number defines a 64-bit double precision floating point number, also following the IEEE standard. The single precision and double precision floating point numbers are identical to the single and double floating point types in most modern programming languages. You can type in floating point numbers using scientific notation. E.g. 1.5e6 is one and a half million. Positive infinity, negative infinity and "not a number" are represented by Inf, -Inf and NaN respectively.

The 6-byte floating point number is a bit of an odd-ball. It defines a 48-bit floating point number which corresponds with the "Real" type in Turbo Pascal and other Pascal variants. In Delphi, the "Real" type is actually a 64-bit double precision floating point number. The "Real48" type is the Delphi equivalent of the Real type in Turbo Pascal.

The little and big endian options switch the order of the bytes. The decimal value 300 is 0x12C in hexadecimal. In little endian format, the two bytes are ordered 2C 01, starting with the least significant byte. In big endian, the two bytes are ordered 01 2C, starting with the most significant byte. You can see the difference if you type 300 in the "word" field in EditPad Pro. The Intel and AMD processors that Windows runs on all use the little endian format natively. The PowerPC processors that the Apple Macintosh used to run on use the big endian format natively. Recent Macs use little endian Intel processors.

## View|Split Editor

Click on the Split Editor item in the View menu to split the editor into two, giving you two independent views of the active file. In the submenu of the Split Editor item you can choose whether the editor should be split horizontally or vertically, or whether the second editor should be placed in a floating window. The floating window is useful if your PC has multiple monitors and you want the second view to be on another monitor. Clicking the Split Editor item directly toggles between not splitting and splitting using the method you last selected in the submenu.

When you switch between files while the editor is split, the editor remains split in the same way, and both halves of the editor will show the file that you switched to. EditPad cannot show more than one file at the same time in a single EditPad instance. Use View|New Editor to start a second instance.

By default, the two halves of the split editor scroll independently. Scrolling one half of the editor does not scroll the other half. If you want both halves to respond to your scrolling simultaneously, use the View|Joint Scrolling command.

Both editor halves maintain their own text cursor position and their own selection. Selecting text in one half does not select it in the other half. If you make a different selection in both halves, you can use Block|Swap Selections to swap the two blocks of text that you selected.

Both editor halves do show the same file. Any changes you make to the file in one half automatically show up in the other half. Only the selections, cursor positions, and scrolling positions are independent.

## View|Joint Scrolling

If you split the editor, then by default the two halves of the split editor scroll independently. This makes it easy to copy and paste between different parts of the same file, or to look at one part of the file while typing into another part of the file.

If the two parts of the file you are working consist of (roughly) the same number of lines but are too long to fit on the screen, select Joint with Fixed Gap in the Joint Scrolling submenu of the View menu. From then on, scrolling one half of the editor automatically scrolls the other half an equal number of lines. EditPad Pro remembers the gap between the line numbers of the first line that was visible in each half of the editor when you selected the Joint with Fixed Gap joint scrolling option. If you scroll to the top or bottom of the file, EditPad Pro may not be able to maintain the gap. But it will still remember the gap. If you scroll back to the middle of the file, the scrolling gap between the two halves will be restored.

If you are concentrating on only one part of a file, and that part is longer than fits on the screen, you can split the editor vertically and then select Joint without Gap in the Joint Scrolling submenu. Then the right hand half of the editor will show the lines immediately following the lines shown in the left hand half. Essentially you get two newspaper columns of your file. This mode is particularly useful when you have a wide screen monitor and the lines in your file are short, or you prefer your lines to be word wrapped to a short length, making them easier to read and edit. This view is maintained when you scroll either half of the editor. If you scroll the left half to the bottom or the right half to the top, then both halves will show the bottom or top of the file. Once you scroll back to the middle of the file, the right half will again show the lines immediately following the left half.

If you want to turn off joint scrolling, either click the Joint Scrolling item in the View menu directly, or select Independent in the submenu. Clicking the item directly toggles between independent scrolling and the previously selected joint scrolling mode.

The Joint with Fixed Gap, Joint without Gap, and Independent options only affect the active file. When you switch between files, the view remains split in the same way, but joint scrolling reverts to what it was for the file you're switching to. If you want to use the same scrolling method for all files, select Joint without Gap for All Files or Independent for All Files in the Joint Scrolling submenu. Then the scrolling method won't change when you switch between files. The Joint with Fixed Gap option does not have an "all files" equivalent. The scrolling gap is always remembered independently for each file.

## View|New Editor

When you try to start a new instance of EditPad through the Windows Start menu or a desktop shortcut, EditPad will activate the instance that was already running instead of starting a new instance. Any files that the new instance should have opened will be opened by the existing instance. Since EditPad is designed to make it easy to work with large numbers of files, you can easily work with all your documents in one location, without cluttering your desktop.

If you do want to have multiple instances of EditPad running, simply select the New Editor item in the View menu. This can be useful if you want to view two files side by side.

In the submenu of the New Editor item, you can select how the two instance should be positioned relative to each other. Clicking the New Editor item directly opens the instance using the option you selected last time.

- On top: Opens the new instance at the same size and position as the current instance.
- Horizontal split: The new instance takes up the bottom half of the area occupied by the current instance. The current instance shrinks to take up the top half of its area.
- Vertical split: The new instance takes up the right hand half of the area occupied by the current instance. The current instance shrinks to take up the left hand half of its area.
- Cascade: Opens the new instance at the same size as the current instance, but positioned a bit lower and a bit more to the right.
- Second monitor: Opens the new instance at the same size as the current instance, but positioned on the next monitor. If you have more than two monitors, repeating this command as many times as you have monitors minus one (to account for the running instance) will put an EditPad instance on each of your monitors.

The Open Active File item in the New Editor submenu has no immediate effect other than toggling the check mark next to the item. When checked, starting a new editor makes the new instance open the file that is active in the current instance.

The Horizontal Split and Vertical Split options change the size of the current instance. If you don't resize the instance that was split after you launched the new instance, then closing the new instance automatically makes the split instance resize itself again to take back the space it occupied before it was split in half by the new instance.

If you want to start a second EditPad instance from a shortcut, script or application, you can do so with the `/newinstance` command line parameter. Then the new instance opens all the files you specify on the command line. It will not affect the size or position of any running instances, or set its own position based on any running instances. You can use an additional parameter such as `/brl100t200r300b400` to set the new instance's bounding rectangle to left 100, top 200, right 300 and bottom 400, counting pixels from the top left corner of the screen.

## View|Other Editor Joint Scrolling

You can use the Other Editor Joint Scrolling command in the View menu whenever you have multiple instances of EditPad Pro running. It works whether you started them with the View|New Editor menu item or via the `/newinstance` command line parameter. Clicking the Other Editor Joint Scrolling item has no immediate effect other than putting a box around its icon to indicate it is active.

If you turn on Other Editor Joint Scrolling in the EditPad Pro instance that has keyboard focus, then it will broadcast a scrolling message to all other EditPad Pro instances whenever you scroll the active file. If Other Editor Joint Scrolling is turned on in any other running EditPad Pro instance, then those instances will scroll their active files to the same line whenever they receive the scrolling messages. The result is that if you turn on Other Editor Joint Scrolling in two or more EditPad Pro instances, then all those instances will scroll simultaneously. This makes it easy to compare two or more files side by side.

## View|Browser

Select Browser in the View menu to open the active file in a web browser. If the file is modified, EditPad will save it first.

If this feature does not work, EditPad is either unable to detect your computer's default web browser, or you've incorrectly configured the browser setting in the System Preferences.

## View|Close Panels

Press the Esc key on the keyboard to close the active side panels. The active panel is the one that has keyboard focus. When the panel is closed, keyboard focus is automatically given to the main editor.

If none of the open panels has keyboard focus, all of them will be closed. To quickly close all panels at any time, simply press the Esc key twice.

## View|Focus Editor

The keyboard shortcut associated with the Focus Editor item in the View menu is a quick way to set the keyboard focus to the main editor. Doing so will make the text cursor (blinking vertical bar) visible in the main editor allowing you to continue editing the file.

Using the Focus Editor keyboard shortcut allows you to quickly switch back and forth between the main editor and other panes. E.g. while working with the search and replace pane, you can press Tab on the keyboard to cycle through the various controls on the search and replace pane. To switch to the main editor, press F9. To switch back to the search and replace pane, press Ctrl+F.

## View|Office 2003 Display Style

Toggles the looks of the toolbars and side panels between the Office 2003 display style, and the current Windows theme. The current Windows theme is the theme you selected in the Control Panel in Windows.

## View|Restore Default Layout

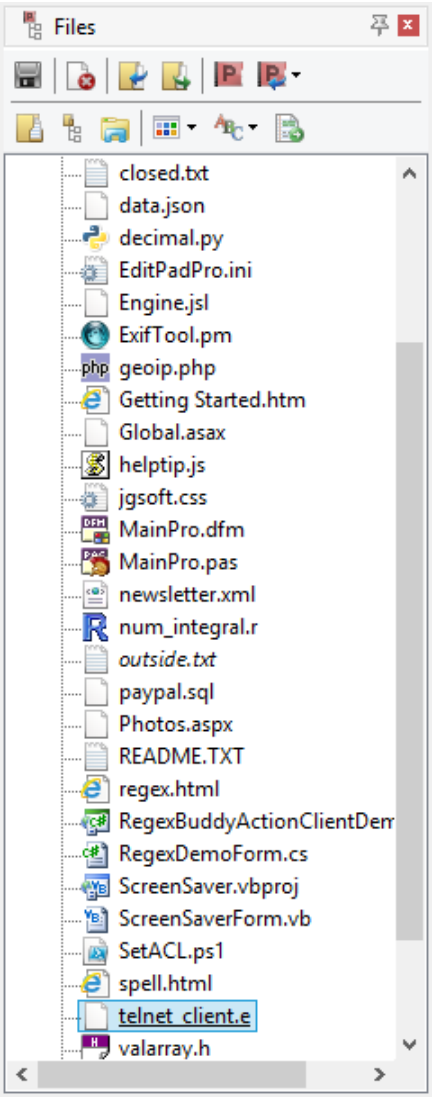
The Restore Default Layout item in the View menu restores all side panels, all toolbars, all context menus, and the main menu to their default configurations and locations. All menu and toolbar customizations you made are undone.

## View|Custom Layouts

If you have spent time rearranging EditPad's toolbars and/or side panels, or if you have customized the toolbars or menus, you should use the Custom Layouts item in the View menu to save those settings. Simply select Save Layouts and type in a name. That name will then appear under the Custom Layouts submenu. If you click on the name in the Custom Layouts submenu, EditPad restores all side panels, all toolbars, all context menus, and the main menu to the configurations and locations they had when you saved the layout.

You can save as many layouts as you like. If you want to update a layout, save it again under the same name. To delete a layout, select the Delete Layout command and then select the layout you want to delete.

# 17. View|Files Panel




The Files Panel is a panel that sits docked at the left side of EditPad's window. You can make it visible by selecting Files Panel in the View menu. You can dock the panel elsewhere by dragging its caption bar or its tab.

The Files Panel shows a collapsible tree of the projects and files you have open in EditPad Pro. All project nodes in the tree sit under the EditPad Workspace root node. Each project node contains nodes for the files in the project. Depending on the options you enabled under the View button (see below), the file nodes may be grouped under folder and file type nodes under project nodes too.

Files that have unsaved changes are highlighted in bold. Closed files that are still part of managed projects are dimmed if you choose to show them at all. Outside files in managed projects are shown in italics.

The Files Panel offers several commands that you can execute via the panel's toolbar, or via the right-click menu. The key difference between these commands and their equivalents that you find in EditPad Pro's main menu is that these commands work on the files and projects that you have selected in the Files Panel, rather than the active project or active file.

### Edit

Double-click on a file node to activate the file for editing. If the file's project isn't active yet, it will be activated too. Double-click on a project node to activate the project, and the file last active in that project.

### Save

If you've selected one or more files, those files will be saved just like File|Save would do. If you've selected a project, that project is saved like Project|Save Project As would do. Saving a project does not save the files in that project. If you've selected a folder, all files listed under that folder in the Files Panel are saved.

### Close

Closes the selected files or projects like File|Close and Project|Close Project would do. Closing files does not remove them from managed projects. If you've selected a folder, all the files listed under that folder are closed, and the folder will disappear from the Files Panel.

### Remove from Project

Closes the selected files and removes them from managed projects like Project|Remove From Project would do. If you've selected a folder, all the files listed under that folder are removed from the project, and the folder will disappear from the Files Panel.

If a project is selected, the project is simply closed as Project|Close Project would do.

### Save Copy As

If you selected one file or one project, you will be asked in which folder and under which name you want to save the copy of the file or project. This works just like File|Save Copy As and Project|Save Copy of Project As do for the current file or project.

If you selected several files, you will be asked for a folder to copy the files into. All selected files will be copied into that folder under the names they already have. This way you can quickly copy a bunch of files.

If you selected several projects, you can choose whether to copy the `.epp` files into which the projects themselves are saved, or whether you want to copy the document files held by the projects, or both. All those files will be copied into the folder you select under the names they already have.

### Rename and Move

If you selected one file or one project, you will be asked for a new name and folder for that file or project, just like the File|Rename and Move and Project|Rename and Move Project do for the active file or project.

If you selected several files, you will be asked for a folder to move the files into. All selected files will be moved into that folder, without being renamed. This way you can quickly move a bunch of files to a different location.

If you selected several projects, you can choose whether to move the `.epp` files into which the projects themselves are saved, or whether you want to move the document files held by the projects, or both. All those files will be moved into the folder you select under the names they already have.

### New Project

Creates a new untitled project. If one or more files are selected, they will be moved into the newly created project. This is a quick way to split a project with many files into multiple projects with a manageable number of files. If no files are selected, the new project will have one untitled blank file.

### Move to Project

Removes the selected files from the project(s) they're in, and adds them to the project that you select in the Move to Project submenu. You need to open the project you want to move the files into before you can move them. The submenu only lists open projects.

This command does not move the files on disk. It only changes which project the files are part of. If you move closed files or outside files from one managed project to another, those files remain closed files or outside files in the project you've moved them to.

### Add Outside Files

If you have one or more outside files selected in a managed project, those files are made part of the project as open files.

### Open Files from This Folder

Opens the File|Open dialog showing the folder that you selected in the Files Panel, or the folder containing the file or project that you selected in the Files Panel. Other than the default folder for the file selection dialog, there is no difference between this command and File|Open.

### Explore in EditPad

Opens the Explorer Panel and selects the folder that you selected in the Files Panel, or the folder containing the file or project that you selected in the Files Panel.

### Explore in Windows Explorer

Launches Windows Explorer showing the folder that you selected in the Files Panel. If you selected a file or a project, then Windows Explorer is launched showing the folder containing that file or project, with that file or project selected in Windows Explorer.

### View

The View button on the Files Panel toolbar has a submenu with options that determine what the Files Panel displays and how the Files Panel is organized.

- Closed Files in Managed Projects: Turn on to make the Files Panel show closed files that are still part of managed projects. These will appear dimmer than files that are open. Turn off to hide all closed files, even if they are still part of managed projects. Files that were closed from unmanaged projects are always removed from the project, and never show up on the Files Panel.
- Collapse Automatically: Turn on to collapse all nodes in the tree on the Files Panel, except those containing the active file. Turn off to expand all nodes, except those that you collapsed manually. If you manually expand a node, that node will stay expanded regardless of this option.
- Flat List of Files: Turn on to show a flat list of files directly under the project nodes, without any folder nodes, even if the files are in different folders. Turn off to show folder nodes under the project nodes to indicate the folder hierarchy that the files are in.
- Group by File Type: Turn on to put nodes for all the file types that are used in the project under the project node. The file and/or folder nodes are then put under the file type nodes. If you turn this off, the Files Panel does not add nodes for file types at all.

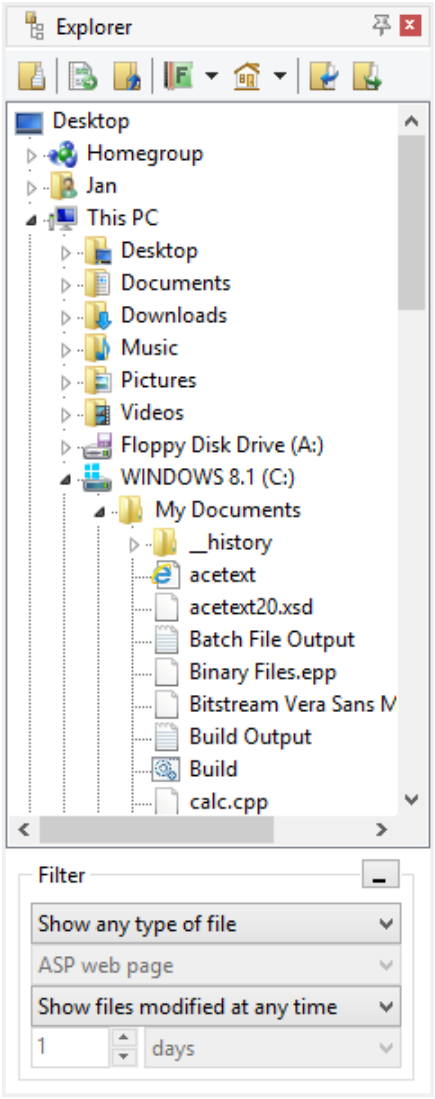
- Paths Relative to Projects: Turn on to add the nodes directly under the project node for files that are in the same folder as the `.epp` project file. If the project has files stored in subfolders of the folder containing the `.epp` file, then only nodes for the subfolders are added. For all other files, a drive letter node is added under the project node and the files and folder nodes are added to the drive letter node. If you turn off this option, drive letter nodes and folder nodes for the full path are added for all files in the Project. This option is not available when showing a flat list of files.
- Subfolders first: Turn on to first list subfolders under project and folder nodes, and then file nodes under the same project and folder nodes. Turn off to list files first, then subfolders. This option is not available when showing a flat list of files.

### Sort

When there are multiple files under the same project node or folder node, the sort option determines how the files are ordered. The option does not affect the order of any other nodes.

- Alphabetically: Sort files alphabetically by their file names.
- Tab Order: List files in the same order as they have in the row of file tabs. This order is determined by the order in which you open files, and which file was active when you opened more files. Dragging and dropping tabs or Files Panel nodes changes the order of both the tabs and the Files Panel nodes.
- Time Last Edited: List files in the order which you last edited them, from most recently edited to least recently edited. If you edit a file in EditPad, it is moved to the top of the list immediately. All files that you have edited in EditPad since you last opened them are ordered by the moment of the last change, regardless of whether those changes were saved. Undoing an action is also considered a change and will move the file to the top of the list. Files that you have opened but not edited in EditPad are ordered by the last modification time stamp the files have on disk.

# 18. View|Explorer Panel




The Explorer Panel is a panel that sits docked at the left side of EditPad's window. You can make it visible by selecting Explorer Panel in the View menu. You can dock the panel elsewhere by dragging its caption bar or its tab.

The Explorer Panel functions as a miniature version of Windows Explorer, the file manager that is part of Windows. With EditPad Pro's Explorer panel, you can easily open files without having to go through File|Open. You can also copy, move and delete files, even if you didn't open those files in EditPad Pro. The copy, move and delete commands in the Explorer Panel work on the files that you've selected in the Explorer panel, rather than the files you have opened in EditPad Pro.

By default, the Explorer Panel shows a tree of all drives, folders and files on your computer. Since the list of files may be too long to work comfortably, you can apply various filters to make the Explorer Panel show only those files you're interested in.

## Open

If you've selected one or more files, the Open button will open those files into the current project, just like File|Open would do. If you've selected one or more folders, the Open button will open all files in all the selected folders and all the subfolders of the selected folders. However, only files passing the Explorer Panel's filter will be opened, since those are the only files visible in the Explorer Panel.

## Save into Folder

This command is identical to the File|Save As command, except that it defaults to the folder that you have selected in the Explorer Panel, or the folder containing the file that you have selected. The file that is saved is the file that you are editing, just like File|Save As does.

## Create Folder

Create a new folder under the folder that you have selected in the Explorer Panel. You will be prompted for the folder's name.

## Select Active File

If the file you're editing in EditPad Pro has a file name (i.e. you've saved it), you can click the Select Active File button to select that file in the Explorer Panel. Drives and folders in the Explorer Panel's tree will be

expanded as needed. If the file you're editing is not visible in the Explorer Panel because you've filtered it out, then the folder containing the file will be selected instead.

### Up Folder

The Up Folder button is a quick way to select the folder that contains the currently selected file or folder. If the home folder is selected when you click the Up Folder button, EditPad Pro will make the home folder's parent folder the new home folder. If the home folder is a drive, then the desktop space becomes the new home folder.

### Favorite Folders

To add a folder to your favorite folders, click on the downward pointing arrow next to the Favorite Folders button, and click on the Add Selected Folder item. You can then quickly select that folder in the Explorer Panel by clicking on the downward pointing arrow next to the Favorite Folders button, and clicking on the folder you want to select.

### Set Home Folder

Click the Set Home Folder button to make the currently selected folder the root folder of the Explorer Panel. Any folders above the home folder will be invisible. If you save all your text files on a particular drive or in a particular folder, setting that drive or folder as the home folder will make the Explorer Panel easier to navigate. EditPad Pro will remember your home folder next time you run it. The Explorer Panel will then load much faster, as it only has to scan the home folder that you specified, rather than all drives and network resources connected to your PC.

### Explore in Windows Explorer

Launches Windows Explorer showing the folder that you selected in the Explorer Panel. If you selected a file or a project, then Windows Explorer is launched showing the folder containing that file or project, with that file or project selected in Windows Explorer.

### Copy

If you selected one file, the Copy button will ask you for a folder where to save the file, and a name to save it under. With one file, the Copy command works like File|Save Copy As, except that it copies the file you selected in the Explorer Panel rather than the active file.

If you selected more than one file, or you selected one or more folders, the Copy button will ask you for a folder to copy all the selected files and folders into. Folders will be copied entirely, including all their files and all their subfolders. All files will be copied, including files that are invisible in the Explorer Panel because of the file filter.

### Rename and Move

If you selected one file, the Rename and Move button will ask you for the folder you want to move the file into, and the new name you want to give it. With one file, the Rename and Move command works like File|Rename and Move, except that it renames and/or moves the file you selected in the Explorer Panel rather than the active file.

If you selected more than one file, or you selected one or more folders, the Rename and Move button will ask you for a folder to move all the selected files and folders into. The files and folders will be moved only, not renamed. Folders will be moved entirely, including all their files and all their subfolders. All files in the selected folders will be moved, including files that are invisible in the Explorer Panel because of the file filter.

### Delete

Click the Delete button to delete the selected files and folders. When deleting files and folders from the hard disk, they will be moved into the Windows Recycle Bin.

### Refresh

Reload the list of files and folders under the selected folder.

# 19. File Filter

EditPad Pro's Explorer Panel and FTP Panel allow you to filter the files they display. Showing only the files you're interested in makes it easier to work with folders containing large numbers of files.

The Explorer and FTP panels are not updated instantly while you edit the filters. To update the panel, either click on it with the mouse or press Enter while one of the filter drop-down lists has keyboard focus.

You can filter files based on their names in four ways:

- Show any type of file: Do not filter files based on their names.
- Show files of type: Shows the files that match the file mask of the file types you've configured. Select the file type you want from the drop-down list.
- Files matching a file mask: Shows the files that match the file mask you type into the drop-down list. You can enter multiple masks delimited by semicolons.
- Show files matching a regex: A file will only be shown if the regular expression you type into the drop-down list matches the file's name. If you enter multiple regular expressions delimited by semicolon, a file will be shown if at least one of the regular expressions matches.

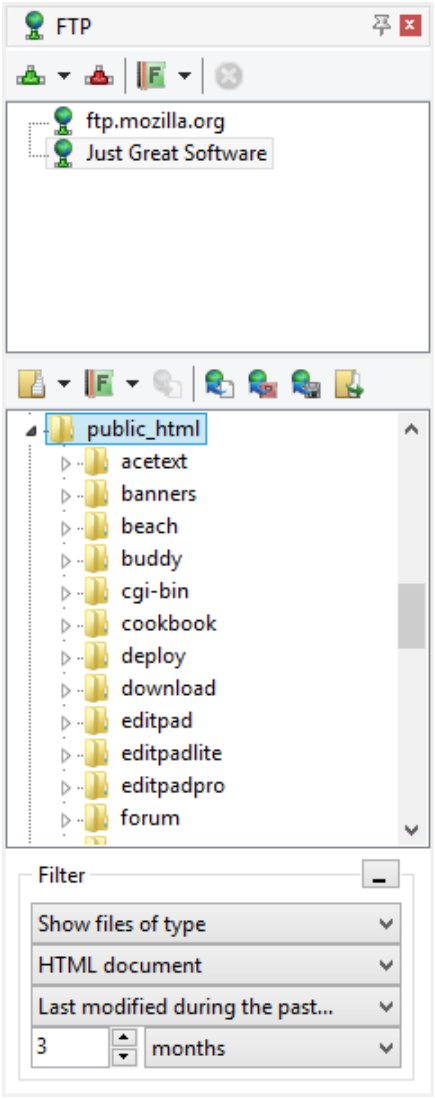
In addition to filtering files based on their names, you can also filter files based on the date and time they were last modified.

- Show files modified at any time: Do not filter files based on their last modification date.
- Last modified during the past...: Only show files that have been last modified in a certain number of past hours, days, weeks, months or years.
- Not last modified during the past...: Only show files that were not last modified in a certain number of past hours, days, weeks, months or years.
- Last modified since...: Only show files last modified on or after a specific date. Files last modified on the date you specify are shown.
- Not last modified since...: Only show files that were last modified before a specific date. Files last modified on the date you specify are shown.

When specifying a time period, EditPad Pro starts counting from the start of the current period. E.g. if you tell EditPad Pro to show files modified during the last two hours at half past three, EditPad Pro will show files modified at or after one o'clock, two hours before the start of the current hour.

Weeks start on Monday. If you tell EditPad Pro to show files modified during the last week on Wednesday the 14th, EditPad Pro will show files modified on or after Monday the 5th. The only exception to this rule is when you filter the files by a number of weeks on a Sunday. Then, EditPad Pro starts counting from the next Monday. The same filter on Sunday the 18th will have EditPad Pro show files modified on or after Monday the 12th.

# 20. View|FTP Panel




The FTP Panel is a panel that sits docked at the left side of EditPad's window. You can make it visible by selecting FTP Panel in the View menu. You can dock the panel elsewhere by dragging its caption bar or its tab.

## FTP Panel Layout

The tree view at the top of the FTP Panel shows all the FTP servers EditPad Pro is currently connected to. EditPad Pro can be connected to any number of FTP servers at the same time. You can transfer files between your PC and all the connected servers simultaneously. Obviously, this will only be practical with a high-speed Internet connection. The tree at the top also shows an upload and download queue for each server below the server's node. You can resize the server tree by dragging the splitter bar immediately below the server tree, above the toolbar with the open, download and upload buttons.

The tree at the bottom shows the folder structure of the FTP server you selected in the list of servers at the top. If you click on another server, the folder view will be replaced instantly. You can do so at any time, even when EditPad Pro is still retrieving part of the folder structure. The retrieval will continue in the background.

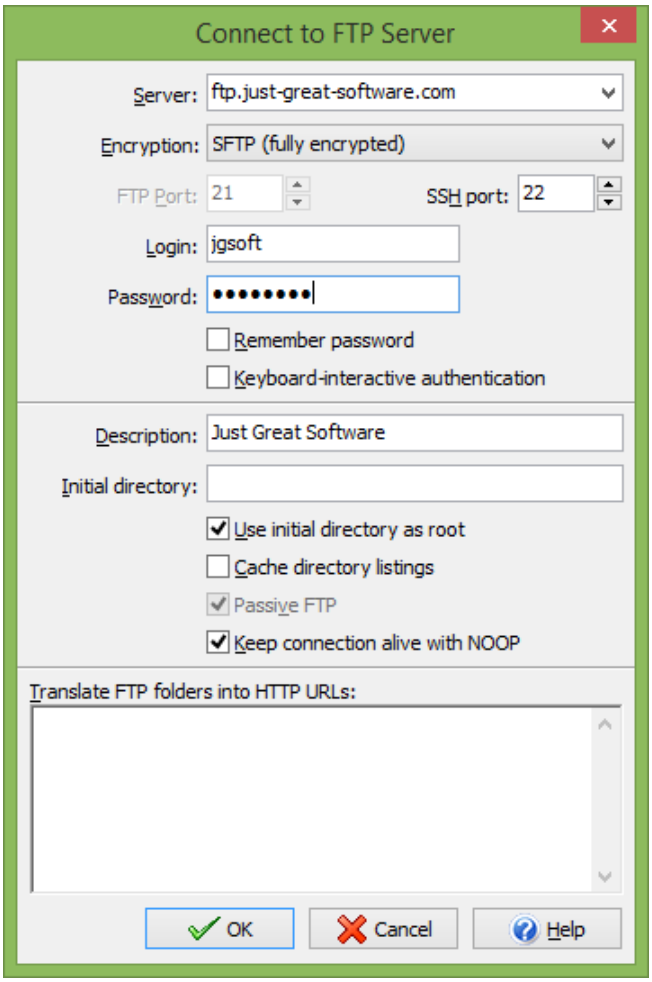
By default, the FTP Panel shows all folders and files on the selected FTP server. Since the list of files may be too long to work comfortably, you can apply various filters to make the FTP Panel show only those files you're interested in.

## ASCII and Binary Mode

Many FTP client have a switch that lets you choose whether to upload and download files in ASCII or binary mode. In binary mode, the FTP client uploads or downloads an identical copy of the file. In ASCII mode, the FTP client and server will attempt to convert the file between the client and server operating systems. E.g. when using a Windows FTP client to upload a text file to a Linux server, Windows-style line breaks will be converted to UNIX-style line breaks when uploading a file, and the other way around when downloading a file.

EditPad Pro always uploads and downloads files in binary mode. You'll always get an exact copy of what's on the server, and you'll always put an exact copy of the file back onto the server. Unlike many other applications, EditPad Pro transparently handles Windows, UNIX and Macintosh text files. Still, you should be careful when uploading files to Linux or UNIX servers. E.g. a CGI script may abort with a mysterious 500 server error if you upload it with Windows-style line breaks onto a web server running Linux. The solution is to select Convert|To UNIX (LF only) in EditPad Pro's menu before uploading the file. You can also save the file on your Windows PC with UNIX-style line breaks, so you need to do the conversion only once. You can also set the default for new files in the File Type Encoding settings.

## Connect to FTP




Click the Connect to FTP button to connect to an FTP server. Enter the server's host name (e.g. ftp.mydomain.com), port, your login or user name and your password. You can also choose whether EditPad Pro should remember your password. EditPad Pro supports all the popular methods for securing FTP connections. You can select the one your server uses from the Encryption drop-down list. If you're not sure which method your server uses, the port number is often a good indication.

- No encryption: plain old unsecure FTP. The default port is 21.
- TLS or SSL, if available: choose this option to easily connect to an FTP server without worrying about encryption, while still using whatever encryption is available. The connection starts with a plain old unsecure connection on port 21. Before logging in, negotiate with the server for TLS and SSL encryption. If the server supports TLS or SSL, the connection will be secured before your login and password are sent. File transfers will be encrypted if the server supports encrypted file transfers. If the server supports TLS or SSL but not encrypted file transfers, your login and password will be secure, but file transfers will be unsecure. If the server does not support TLS or SSL negotiation, the FTP session will continue completely unsecured.
- TLS, fully encrypted: this is the recommended method to require a fully encrypted FTP connection. The connection starts with a plain old unsecure connection on port 21. Before logging in, request TLS encryption from the server. If the server supports TLS, the connection will be secured before your login and password are sent. If the server does not support TLS, the FTP session is aborted and EditPad will show an error message. EditPad Pro will encrypt file transfers. If the server does not support encrypted file transfers, EditPad Pro will be able to connect to the server but will then fail to retrieve any directory listings and will fail to transfer any files.
- TLS, files unencrypted: use this method if your server supports TLS, but not encrypted file transfers, which is fairly common. THe connection starts with a plain old unsecure connection on port 21. Before logging in, request TLS encryption from the server. If the server supports TLS, the connection will be secured before your login and password are sent. If the server does not support TLS, the FTP session is aborted and EditPad will show an error message. EditPad Pro will not encrypt file transfers. If the server does not allow unencrypted file transfers, EditPad Pro will be able to connect to the server but will then fail to retrieve any directory listings and will fail to transfer any files.
- SSL, fully encrypted: use this method if your FTP server supports encryption, but does not support negotiating for the encryption. EditPad will establish an FTP connection encrypted with SSL. The default port is 990. If the server does not support SSL, the FTP session is aborted and EditPad will

show an error message. EditPad Pro will encrypt file transfers. If the server does not support encrypted file transfers, EditPad Pro will be able to connect to the server but will then fail to retrieve any directory listings and will fail to transfer any files.

- SSL, files unencrypted: use this method if your FTP server supports non-negotiated encryption, but does not support encrypted file transfers. EditPad will establish an FTP connection encrypted with SSL. The default port is 990. If the server does not support SSL, the FTP session is aborted and EditPad will show an error message. EditPad Pro will not encrypt file transfers. If the server does not allow unencrypted file transfers, EditPad Pro will be able to connect to the server but will then fail to retrieve any directory listings and will fail to transfer any files.
- SFTP (fully encrypted): this is the recommended method for SSH servers. EditPad Pro will connect via SSH and use the SFTP protocol to transfer files directly via the SSH connection. If your server supports SSH, the SFTP option is most likely what you need. The default port is 22. SFTP connections are always fully encrypted. The FTP protocol is not used at all. (SFTP and FTP are totally unrelated, even though they are similarly named and serve a similar purpose. EditPad Pro handles the differences transparently.)
- SSH (fully encrypted): this is an alternative method for servers that run both SSH and FTP but that didn't configure SFTP in the same way as FTP. EditPad Pro will open an SSH (secure shell) connection, usually on port 22, and use it as a secure tunnel to connect to the FTP server running on the same host, usually on port 21. The server must accept SSH connections from your PC, and unsecure FTP connections from localhost (i.e. the server itself). The SSH tunnel will fully encrypt the entire FTP session.

To connect to an FTP server, you need to enter a username and password. To connect to an SSH server, you need to enter a username and either enter a password or tick the "keyboard-interactive authentication" checkbox. That depends on whether your SSH server requires password authentication or keyboard-interactive authentication. Keyboard-interactive authentication will result in a prompt during the connection. Usually this prompt asks for a password. The actual prompt depends on what the server asks from EditPad.

Optionally, you can also enter a description for the FTP server. This description will appear in history lists instead of the default username@hostname. The initial directory is the directory that EditPad Pro navigates to when you first connect to the FTP server. If you don't specify an initial directory, EditPad Pro will let the FTP server choose the initial directory. If you turn on the option "use initial directory as root", you will only be able to access the initial directory and its subdirectories. You won't be able to access the directory's parent or sibling directories. This option is available whether you specified the initial directory or not. If you have a slow internet connection, you can tell EditPad Pro to keep a cached copy of the FTP server's directory listings. This will speed up future connections to the same server. You may want to turn off this feature if you often modify files on the FTP server outside of EditPad Pro, as that will cause the cached listings to be out of date, which may be confusing. If EditPad Pro cannot retrieve an FTP server's directory listings, try toggling the "passive FTP" option. While passive FTP is usually preferred, not all servers support it. If a server times out the connection quickly, you can tell EditPad Pro to send "no operation (NOOP)" commands to the server every 30 seconds. Not all servers will reset the time-out in response to NOOP commands.

In the "translate FTP folders into HTTP URLs" box, you can enter a list of FTP folder paths and their corresponding HTTP URLs. Enter one folder=URL pair per line, separating the folder and URL with an equals sign. The folder must be a full path. When you select the View|Browser command when you're editing a file in that folder, EditPad Pro will replace the folder path in the file's path with the URL. E.g. if you specify /usr/home/me/public_html/=http://www.mydomain.com and you're editing the file /usr/home/me/public_html/subfolder/file.html then EditPad Pro will open the URL http://www.mydomain.com/subfolder/file.html

When you want to connect to the same server again, you can select it from the Connect button's drop-down menu. If the password was remembered, EditPad Pro will connect to the server instantly. Otherwise, it will ask you for the password.

You can connect to as many servers as you like. Connecting to another server does not break previous connections. Simply click on a connection in the top half of the FTP panel to see the file listings. You can also connect to the same server more than once if you use a different login name. Since a server may restrict the number of connections from a single computer, you may have to disconnect from the server before reconnecting with a different login. EditPad Pro does not impose any restrictions on the number of simultaneous connections to an FTP server.

### Disconnect from FTP

Click the Disconnect from FTP button to break EditPad Pro's connection with the selected FTP server. Any pending uploads or downloads will be aborted. The server will be removed from the list of servers in the top half of the FTP panel.

If the FTP server or a network outage breaks the connection, the server will remain listed in the FTP panel. EditPad Pro won't notice that the connection was broken until you try to upload or download another file. When you do, EditPad Pro will automatically try to reconnect to the server every 10 seconds until it either succeeds, or you click the Disconnect button.

### Favorite Servers

The Connect to FTP button will automatically keep a history of the last 16 FTP servers that you connected to. If you connect to lots of FTP servers, and use some more frequently than others, you can add the FTP servers you use frequently to your list of favorite FTP servers. To add a server to the list, connect to it. Then click on the downward pointing arrow next to the Favorites button, and click on the Add Connected FTP Server item. To connect to a favorite FTP server, select it from the Favorites button's drop-down menu.

### Abort

Aborts all uploads and downloads to and from the selected server that are currently pending or in progress.

### Open from FTP

Click the Open from FTP button to open all selected files for editing. EditPad Pro will instantly create new, empty tabs for the files. The files are added to the FTP server's download queue. While a file downloads, its text will appear in EditPad Pro bit by bit. You can start editing the downloaded portion right away. The remainder of the file will be appended as it is downloaded. If you close a tab before the file is downloaded, that file's download will be aborted.

You cannot open entire folders from FTP like you can open entire folders in the Explorer Panel. If you click Open from FTP with a folder selected, that folder's node will simply be expanded. EditPad Pro will retrieve the list of subfolders and files, but won't actually download the files themselves.

EditPad Pro will remember that you opened the files from FTP. When you use File|Save or File|Save All, or you confirm the save question asked by File|Close and similar commands, EditPad Pro will automatically upload the file back to the FTP server, overwriting the original file. To save the file on your computer, use File|Save As instead. Once you save the file on your own computer, EditPad Pro will *not* automatically upload it to the FTP server when you save.

The drop-down menu of the open button shows a list of files that you have recently opened from this server. Select a file from the drop-down menu to open it again.

### Favorite Files

If you often edit particular files on this server, you can add them to the list of favorite files via the Favorites button next to the Open button. EditPad Pro keeps a list of favorite files for each server that you added to your favorite servers, and each server that is remembered by the Connect to FTP button. These lists are separate. To open a particular favorite file, first connect to the server the file is on, and then select the file from the favorite files on that server.

### Download

Click the Download button to download the selected files to disk, without opening them for editing in EditPad Pro. If you selected one file, EditPad Pro will ask you where and with which name you want to save it. If you selected more than one file, EditPad Pro will ask you for the folder where you want to save the selected files. All files will be downloaded with the same name as they have on the FTP server.

### Upload File

Click the Upload File button to upload the file you're currently editing to the FTP server. If you selected a folder in the FTP panel, the file will be uploaded into that folder. If you selected a file in the FTP panel, the file you're editing will be uploaded into the same folder as the selected file. EditPad Pro will ask you which name you want the uploaded file to have on the FTP server.

If the file you're uploading was untitled, EditPad Pro will change the tab's caption to the file name you used to upload the file. If you later modify the file and use File|Save or File|Save All, or you confirm the save question asked by File|Close and similar commands, EditPad Pro will automatically upload the file back to the FTP server, overwriting the original file. To save the file on your computer, use File|Save As instead. Once you save the file on your own computer, or if it already had been saved on your own computer before you uploaded it, EditPad Pro will *not* automatically upload it to the FTP server when you save.

### Upload Project

Click the Upload Project button to upload all files in the current project. If you selected a folder in the FTP panel, the files will be uploaded into that folder. If you selected a file in the FTP panel, the project's files will be uploaded into the same folder as the selected file.

If one or more of the files in the project are untitled, EditPad Pro will ask you for a name for those files. Files that were previously uploaded or saved will be uploaded with the name they already have. Files that were

untitled when you uploaded them will be automatically uploaded when you save them, just like when uploading a single untitled file.

### Upload from Disk

Click the Upload Project button to upload files that you aren't editing in EditPad Pro. EditPad Pro will show an open file dialog to select the files to upload. You can select multiple files to be uploaded. If you selected a folder in the FTP panel, the files will be uploaded into that folder. If you selected a file in the FTP panel, the files will be uploaded into the same folder as the selected file.

Should you use the Upload from Disk file to upload a file that you're editing in EditPad Pro, then the copy of the file on disk will be uploaded. If the file has unsaved changes in EditPad Pro, those changes won't appear in the uploaded file.

### Create Folder

Click the Create Folder button to create a new folder on the FTP server. If you selected a folder in the FTP panel, the new folder will become a subfolder of the selected folder. If you selected a file in the FTP panel, the folder will become a subfolder of the folder that holds the selected file.

### Refresh Folder

EditPad Pro automatically updates the contents of a folder shown in the files and folders tree on the FTP panel whenever you use EditPad to upload, rename, or delete a file via FTP in that folder. If you use another way to add or delete files on the FTP server, then EditPad won't show those changes on the FTP panel. To make EditPad Pro check for changes in a particular folder, select that folder or a file in that folder and invoke the Refresh Folder command via the context menu.

If you regularly use another FTP client, you may want to turn off the option to cache directory listings when you connect to the FTP server in EditPad. Then EditPad will refresh all folder listings each time you connect to the FTP server.

### Rename

Rename the selected file or folder. You will be prompted for the new name.

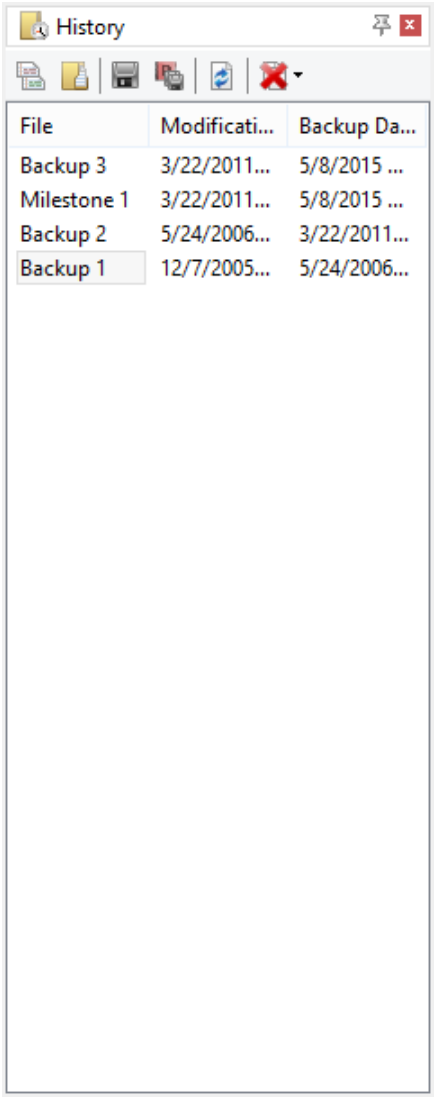
### File Permissions

If your FTP server is running on a UNIX or Linux host, select the File Permissions item in the context menu if you want to change the permissions on the selected file. These are the permissions that you can set via the `chmod` command in UNIX or Linux.

## Delete

To delete a file or folder, right-click on it and select Delete in the context menu. Most FTP servers will not allow you to delete a folder until you've deleted all of its files and subfolders. You can delete multiple files and folders by selecting them all, and then right-clicking on one of the selected items.

# 21. View|File History




The File History is a panel that sits docked at the left side of EditPad's window. You can make it visible by selecting File History in the View menu. You can dock the panel elsewhere by dragging its caption bar or its tab.

The File History shows a list of all backup files that exist for the file you're currently editing in EditPad Pro. To determine which files are backups, the File History uses the "backup copies" setting in the Save Files Preferences. The File History will only show files that follow the naming method you selected in the Save Files Preferences. To make the File History useful, you should select one of the "multi backup" options or the "hidden history" option. These are the only backup methods that can keep more than one backup copy of any given file.

The File History will display two dates for each backup file: the modification date and the backup date. When creating a backup copy, EditPad Pro sets the backup's last modification time stamp to that of the original file. E.g. Today, December 1st, you open a file last modified on November 27th. When you save the file, EditPad Pro will create a backup copy with a modification date of November 27th. The original file will get a modification date of December 1st. When you view the contents of the backup folder in Windows Explorer, the backup file's date will show the November 27 modification date.

The backup date indicates the date that the backup copy was made. In the example above, that would be December 1st. If you right-click on the file in Windows Explorer and select properties, the file's creation date will indicate the date the backup copy was made.

### Compare

Select one of the backup copies and click the Compare button to show the differences between the backup copy and the state the file currently has in EditPad Pro. You will be asked for the same file comparison options used by Extra|Compare Files. The backup copy will be highlighted as the "old" file, and the file you're editing in EditPad pro as the "new" file.

To compare two backup copies with each other, first open the younger copy of the two. Then, select the older copy and click the Compare button. The backup copy you've opened in EditPad Pro will always be highlighted as the "new" file, because it takes the role of the file you're currently editing. Therefore, you should open the younger copy of the two you want to compare. This will avoid confusion as to which is which.

When the active tab has file comparison highlights, the File History will show the full paths of the files that were compared along with their colors. The color for the original file, or the "old" file, is used to highlight

lines only present in that file. The color for the edited file, or the "new" file, is used to highlight lines only present in that file. Lines present in both files are not highlighted.

### Open

Select a backup file and click the Open button to open the file in a tab for viewing or editing. The tab will indicate the file name of the backup file. If you try to save the backup file with File|Save, EditPad Pro will prompt you for a file name. You should save the file under a new name rather than overwriting a backup file. If you overwrite a backup file, EditPad Pro will make a backup of the backup, which can get confusing.

If you open a backup copy via the File History, the File History will continue to display the backup copies of the original file. You should not use File|Open or other commands to open backup copies outside the File History, because then the link between the backup and the original won't be established.

### Save Milestone

The Save Milestone button saves the file you're editing in EditPad Pro. However, it does not overwrite the original file. Instead, it creates a separate "milestone" copy. The milestone copy will appear among the backup files in the File History. The difference between a milestone copy and a backup copy is that EditPad Pro will automatically delete backup copies when they exceed the number and age limits you specified in the Open Files Preferences. Milestone copies are never automatically deleted. They function as a more permanent backup. You should make a milestone copy whenever you're embarking on a more involved editing task that you might want to undo entirely.

If you forgot to save a milestone copy, you can do so afterwards as long as there's still a backup copy of the file in the state it had when you should have saved the milestone. Simply open that backup copy via the File History, and click the Save Milestone button. When you save a milestone of a backup file that you've opened via the File History, EditPad Pro will make a copy of that backup file, and turn it into a milestone of the original file.

If you save a milestone of a file that has unsaved changes, the milestone will be identical to the in-memory copy of the file rather than a copy of the file on disk. The milestone will include your unsaved changes. The file on disk is not affected. So you could use the milestone feature to save unsaved changes without actually saving the file, in order to make a backup of changes you plan to abandon.

### Save Milestone for All Files in Project

The Save Milestone for All Files in Project button will create a milestone copy of each file in the current project, just like the Save Milestone button does for the current file.

### Revert

Click the Revert button to replace the original file with the selected backup file. EditPad Pro will first create a new backup copy from the original file, and then replace the original file with the selected backup file. The last modification date on the reverted file will be that of the backup file, rather than that of the moment you clicked the Revert button. The selected backup file is deleted in the process.

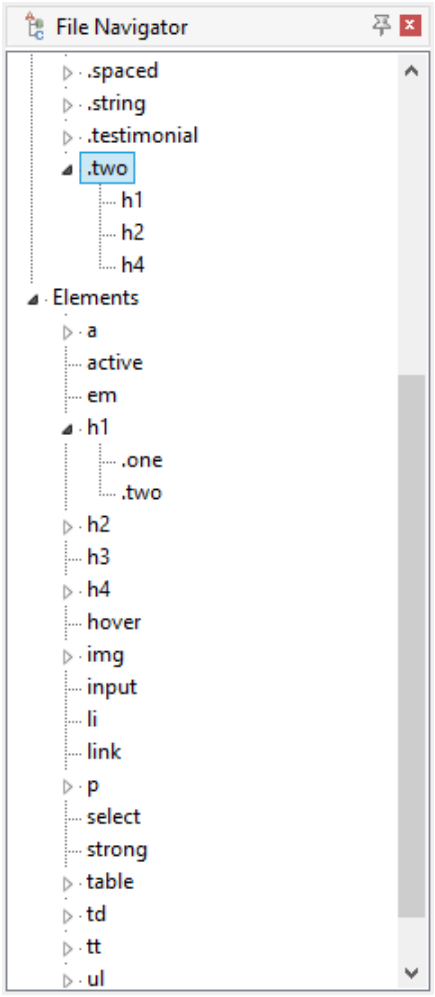
If you change your mind after reverting a file, select the backup file with the most recent backup date, which is the backup created by the reversion. Click the Revert button again to restore the file and its backups to the way they were before the first reversion. If you are using the numbered backups, the numbers of the backups will have shifted, but their modification dates and their contents will be as they were.

If you edit and save a file after reverting it, the backup made when saving will have the same modification date as the backup that you reverted to and will thus appear at that position in the list of backups rather than at the top.

### Delete

The Delete button has 3 subitems. You can choose to delete only the selected backup file, all backup files for the file that is active in EditPad, or all backup files for all files in the project that is active in EditPad.

# 22. View | File Navigator




The File Navigator is a panel that sits docked at the right hand side of EditPad's window. You can make it visible by selecting File Navigator in the View menu. You can dock the panel elsewhere by dragging its caption bar or its tab.

The File Navigator requires a file navigation scheme to be functional. Without such a scheme, the File Navigator will remain empty. You can select a file navigation scheme for each file type on the Navigation tab in the file type configuration.

Based on the file navigation scheme, the File Navigator displays the structure of the file in a collapsible tree. By clicking on items in the tree, you can quickly navigate to various parts of the file. Some items may be linked to more than one part in the file. If you click the same item repeatedly, the file navigator will cycle through all the parts the item is linked to.

When you click an item, the File Navigator will keep keyboard focus. To give the editor keyboard focus, double-click an item or press Enter while the File Navigator has keyboard focus.

To select the part of the file an item is linked to, press the Shift key on the keyboard while you click on the item. If you press Shift and double-click, the part will be selected and the editor will receive keyboard focus. If you prefer to use the keyboard, keep Shift depressed while you navigate with the arrow keys. Press Shift+Enter to select the part and activate the editor.

If an item is linked to multiple parts of the file, those parts become linked in an additional way. If the cursor is already inside one of the parts, you can press Alt+Arrow Up and Alt+Arrow Down on the keyboard to cycle through the linked parts while the editor has keyboard focus. E.g. the Delphi scheme links method declarations from the interface section of a unit and the method implementations from the implementation section into a single node representing the method. When the text cursor is inside a method declaration, you can press Alt+Up or Alt+Down to jump to the implementation, and back.

# 23. Help Menu

## Keyboard Navigation and Editing Shortcuts

The shortcuts in this list are the shortcuts that are recognized by every full text editor control in EditPad, such as the main editor, the search box, the replace box, the clip editor, etc.

Some of these shortcuts are also used by menu items. If you select the command from the menu, it will always work on the main editor. If you select the command by pressing the shortcut key combination, it will work on the editor that has input focus. This is the editor that shows the text cursor, the blinking vertical bar. This is true for all menu items listed below, but not for any other menu items.

All shortcuts that are used by menu items can be configured in Options|Preferences|Keyboard. All other shortcuts cannot be changed.

When a key combination only works in text mode, this is indicated by [text]. If it only works in hexadecimal mode, the indication is [hex].

### Cursor movement keys

Arrow key: Moves the text cursor (blinking vertical bar).

Ctrl+Arrow Left [text]: Moves the text cursor to the start of the previous word or the end of the previous line, whichever is closer.

Ctrl+Arrow Right [text]: Moves the text cursor to the start of the next word or the end of the current line, whichever is closer.

Ctrl+Arrow Up or Down: Scrolls the text one line up or down, or jump to the next paragraph (your preference). When scrolling, the cursor moves along unless it's at the top or bottom (configurable).

Ctrl+Alt+Arrow Up or Down: Moves the text cursor to the previous or next occurrence of the word under the text cursor.

Page Up or Down: Moves the text cursor up or down an entire screen.

Ctrl+Page Up or Down: Scrolls the text one screen up or down.

Home: Moves the text cursor to the beginning of the line. Can be configured to move to the first non-whitespace character of the line, and to the very start of the line after a second press.

Ctrl+Home: Moves the text cursor to the start of the entire text.

End: Moves the text cursor to the end of the line. Can be configured to move to the last non-whitespace character of the line, and to the very end of the line after a second press.

Ctrl+End: Moves the text cursor to the end of the entire text.

Shift+Movement key: Moves the text cursor and expand or shrink the selection towards the new text cursor position. If there was no selection, one is started. Pressing Ctrl as well, will move the text cursor correspondingly.

Alt+Shift+Movement [text]: The same as when Alt is not pressed, except that the selection will be rectangular instead of flowing along with the text.

Any of the above key combinations that do not select part of the text will remove the current selection (the selection, not the selected text) unless selections are persistent. Any key combination that moves the text cursor will also cause the text to scroll to keep the text cursor in view, if the move makes the text cursor invisible. Any key combination that scrolls the text will let the text cursor keep its position relative to the text, unless you chose to keep the text cursor in view while scrolling.

Ctrl+]: Go|Go to Matching Bracket

Ctrl+[: Block|Between Matching Brackets

### Editing commands

Enter: In the main editor: inserts a line break. In the search or replace box: search for the next occurrence.

Shift+Enter: Inserts a line break.

Ctrl+Enter: Edit|Insert Page Break

Delete: Deletes the current selection if there is one and selections are not persistent. Otherwise, the character to the right of the text cursor is deleted.

Ctrl+Delete: Deletes the current selection if there is one. In text mode, if there is no selection, the part of the current word to the right of the text cursor is deleted. If the cursor is not on a word, all characters to the right of the cursor up to the start of the next word are deleted.

Shift+Ctrl+Delete: In text mode, all the text on the current line to the right of the text cursor is deleted. In hexadecimal mode, the selection is deleted.

Backspace: Deletes the current selection if there is one and selections are not persistent. Otherwise, the character to the left of the text cursor is deleted.

Ctrl+Backspace: Deletes the current selection if there is one and selections are not persistent. In text mode, if there is no selection, the part of the current word to the left of the text cursor is deleted. If the cursor is not on a word, all characters to the left of the cursor up to the start of the next word are deleted.

Shift+Ctrl+Backspace: Deletes the current selection if there is one and selections are not persistent. In text mode, if there is no selection, all the text on the current line to the left of the text cursor is deleted [text].

Alt+Backspace: Alternative shortcut for Edit|Undo

Alt+Shift+Backspace: Alternative shortcut for Edit|Redo

Ctrl+Z: Edit|Undo

Ctrl+Y: Edit|Redo

Insert: Toggles between insert and overwrite mode.

Tab: In text mode, if there is a selection, the entire selection is indented. If there is no selection, a tab is inserted. In hexadecimal mode, pressing Tab makes the text cursor switch between the hexadecimal side and text side.

Shift+Tab: In text mode, if there is a selection, the entire selection is unindented (outdented). If there is no selection and there is a tab, or a series of spaces the size of a tab, to the left of the text cursor, that tab or spaces are deleted.

Ctrl+A: Edit|Select All

Shift+F5: Go|Previous Editing Position

Ctrl+Alt+Y: Edit|Delete Line

Shift+Ctrl+Y: Edit|Duplicate Line

Shift+Ctrl+B: Block|Begin Selection

Shift+Ctrl+E: Block|End Selection

Shift+Ctrl+D: Block|Expand Selection

Ctrl+D: Block|Duplicate

Ctrl+M: Block|Move

Ctrl+Alt+B: Block|Go to Beginning

Ctrl+Alt+E: Block|Go to End

Shift+Ctrl+]: Edit|Insert Matching Bracket

### Clipboard commands

Ctrl+X: Edit|Cut

Shift+Ctrl+X: Edit|Cut Append

Ctrl+C: Edit|Copy

Shift+Ctrl+C: Edit|Copy Append

Ctrl+V: Edit|Paste

Shift+Ctrl+V: Edit|Swap with Clipboard

Shift+Delete: Alternative shortcut for Edit|Cut

Ctrl+Insert: Alternative shortcut for Edit|Copy

Shift+Insert: Alternative shortcut for Edit|Paste

## Search Toolbar

Some of the search options on the search toolbar have Alt+letter keyboard shortcuts that are also used by main menu items. When the search toolbar or search panel has keyboard focus, the search options take precedence when you use an Alt+letter shortcut. Otherwise, the main menu takes precedence.

## Characters with Diacritics

While you can use the Character Map to insert any character that you can't type on your keyboard, you can type many characters with diacritics in EditPad even if they don't appear on your keyboard. First, hold down the Ctrl key and press a punctuation key. If your keyboard uses the Shift key to type a particular punctuation character, hold down the Shift key too. Release all keys. Then type in a letter from A to Z, holding down Shift if you want a capital letter.

In EditPad Pro, Ctrl+/ is a shortcut for Block|Toggle Comment by default. If you want to use Ctrl+/ to type characters with strokes, remove the shortcut from Block|Toggle Comment in the keyboard preferences.

| **Punctuation** | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ctrl+'** | á |  | ć | ð | é |  | ǵ |  | í |  | ḱ | ĺ | ḿ | ń | ó | ṕ |  | ŕ | ś |  | ú |  | ẃ |  | ý | ź |
| **Ctrl+`** | à |  |  |  | è |  |  |  | ì |  |  |  |  | ǹ | ò |  |  |  |  |  | ù |  | ẁ |  | ỳ |  |
| **Ctrl+:** | ä |  |  |  | ë |  |  | ẗ | ï |  |  |  |  |  | ö |  |  |  |  | ẗ | ü |  | ẅ | ẍ | ÿ |  |
| **Ctrl+^** | â |  | ĉ | d̂ | ê |  | ĝ | ĥ | î | ĵ | ǩ | l̂ |  | ň | ô |  |  | ř | ŝ | t̂ | û |  | ŵ |  | ŷ | ž |
| **Ctrl+~** | ã |  |  |  | ẽ |  |  |  | ĩ |  |  |  |  | ñ | õ |  |  |  |  |  | ũ | ṽ |  |  | ỹ |  |
| **Ctrl+,** |  |  | ç | ḑ | ȩ |  | ģ | ḫ |  |  | ķ | ļ |  | ṇ |  |  |  | ŗ | ş | ţ |  |  |  |  |  |  |
| **Ctrl+@** | å |  | © |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ® |  |  | ů |  | ẘ |  | ẙ |  |
| **Ctrl+/** |  | ƀ | ¢ | đ | € | ƒ | ǥ | ħ | ɨ |  |  | ł |  |  | ø |  |  |  |  | ŧ |  |  |  |  | ¥ | ƶ |
| **Ctrl+&** | æ |  |  |  |  |  |  |  |  |  |  |  |  |  | œ |  |  |  | ß |  |  |  |  |  |  |  |
| **Punctuation** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z** |
| **Ctrl+'** | Á |  | Ć | Đ | É |  | Ǵ |  | Í |  | Ḱ | Ĺ | Ḿ | Ń | Ó | Ṕ |  | Ŕ | Ś |  | Ú |  | Ẃ |  | Ý | Ź |
| **Ctrl+`** | À |  |  |  | È |  |  |  | Ì |  |  |  |  | Ǹ | Ò |  |  |  |  |  | Ù |  | Ẁ |  | Ỳ |  |
| **Ctrl+:** | Ä |  |  |  | Ë |  |  | Ḧ | Ï |  |  |  |  |  | Ö |  |  |  |  |  | Ü |  | Ẅ | Ẍ | Ÿ |  |

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ctrl+^** | Â | | Ĉ | Ď | Ê | | Ĝ | Ĥ | Î | Ĵ | Ǩ | Ľ | | Ň | Ô | | | Ř | Ŝ | Ť | Û | | Ŵ | | Ŷ | Ž |
| **Ctrl+~** | Ã | | | | Ẽ | | | | Ĩ | | | | | Ñ | Õ | | | | | | Ũ | Ṽ | | | Ỹ | |
| **Ctrl+,** | | | Ç | Ḍ | Ḛ | | Ģ | Ḥ | | | Ḳ | Ḻ | | Ṇ | | | | Ṛ | Ṣ | Ṭ | | | | | | |
| **Ctrl+@** | Å | | © | | | | | | | | | | | | | | | ® | | | Ů | | | | | |
| **Ctrl+/** | | | ¢ | Ð | € | ƒ | Ǥ | Ħ | Ɨ | | | �£ | | | Ø | | | | | Ŧ | | | | | ¥ | Ƶ |
| **Ctrl+&** | Æ | | | | | | | | | | | | | | Œ | | | | § | | | | | | | |
| **Punctuation** | **2** | **3** | **4** | | | | | | | | | | | | | | | | | | | | | | | |
| **Ctrl+/** | ½ | ¾ | ¼ | | | | | | | | | | | | | | | | | | | | | | | |

## Tip of The Day

The first time EditPad starts without any prompts (such as it not being the default text editor), it will show the tip of the day screen at startup. The tips highlight some of EditPad's more interesting features. Click the More Info button to open EditPad's help file to learn more about the feature described in the tip. If you're in the mood for reading tips, click the Next Tip button to read more tips.

Each time you start EditPad it will show a different tip. If you dismiss the tip of the day window within 3 seconds, such as by pressing the Esc key on the keyboard, EditPad assumes you didn't read the tip, and will show the same tip next time. If you want to go back to a tip that you read in the past, click the Previous Tip button.

If you don't want the tip of the day to be shown each time you start EditPad, clear the "show tip of the day at startup" checkbox before you dismiss the tip of the day window. If you want to show the tips again, select Tip of the Day in the Help menu, and tick the checkbox.

## Help | EditPad Web Site

Launches your web browser to open http://www.editpadpro.com/

## Help | Check for New Version

Launches your web browser to show a page that will tell you if the version of EditPad you are currently using, is the most recent one or not.

If a newer version is available, you will be able to download it immediately.

## Share Experiences and Get Help on The User Forums

When you click the Login button you will be asked for a name and email address. The name you enter is what others see when you post a message to the forum. It is polite to enter your real, full name. The forums are private, friendly and spam-free, so there's no need to hide behind a pseudonym. While you can use an anonymous handle, you'll find that people (other EditPad users) are more willing to help you if you let them know who you are. Support staff from Just Great Software will answer technical support questions anyhow.

The email address you enter is be used to email you whenever others participate in one of your discussions. The email address is never displayed to anyone, and is never used for anything other than the automatic notifications. EditPad's forum system does not have a function to respond privately to a message. If you don't want to receive automatic email notifications, there's no need to enter an email address.

If you select "never email replies", you'll never get any email. If you select "email replies to conversations you start", you'll get an email whenever somebody replies to a conversation that you started. If you select "email replies to conversations that you participate in", you'll get an email whenever somebody replies to a conversation that you started or replied to. The From address on the email notifications is forums@jgsoft.com. You can filter the messages based on this address in your email software.

EditPad's forum system uses the standard HTTP protocol which is also used for regular web browsing. If your computer is behind an HTTP proxy, click the Proxy button to configure the proxy connection.

If you prefer to be notified of new messages via an RSS feed instead of email, log in first. After EditPad has connected to the forums, you can click the Feeds button to select RSS feeds that you can add to your favorite feed reader.

## Various Forums

Below the Login button, there is a list where you can select which particular forum you want to participate in. The "EditPad" forum is for discussing anything related to the EditPad software itself. This is the place for technical support questions, feature requests and other feedback regarding the functionality and use of EditPad.

The "regular expressions" forum is for discussing regular expressions in general. Here you can talk about creating regular expressions for particular tasks, and exchange ideas on how to implement regular expressions with whatever application or programming language you work with.

## Searching The Forums

Before starting a new conversation, please check first if there's already a conversation going on about your topic. In the top right corner of the Forum pane, there's an edit box on the toolbar that you can use to search for messages. When you type something into that box, only conversations that include at least one message containing the word or phrase you typed in are shown. The filtering happens in real time as you type in your word or phrase.

Note that you can enter only one search term, which is searched for literally. If you type "find me", only conversations containing the two words "find me" next to each other and in that order are shown. You cannot use boolean operators like "or" or "and". Since the filtering is instant, you can quickly try various keywords.

If you find a conversation about your subject, start with reading all the messages in that conversation. If you have any further comments or questions on that conversation, reply to the existing conversation instead of starting a new one. That way, the thread of the conversation stays together, and others can instantly see what you're talking about. It doesn't matter if the conversation is a year old. If you reply to it, it moves to the top automatically.

## Conversations and Messages

The left hand half of the Forum pane shows two lists. The one at the top shows conversations. The bottom one shows the messages in the selected conversation. You can change the order of the conversations and messages by clicking on the column headers in the lists. A conversation talks about one specific topic. In other forums, a conversation is sometimes called a thread.

If you want to talk about a topic that doesn't have a conversation yet, click the New button to start a new conversation. A new entry appears in the list of conversations with an edit box. Type in a brief subject for your conversation (up to 100 characters) and press Enter. Please write a clear subject such as "scraping an HTML table in Perl" rather than "need help with HTML" or just "help". A clear subject significantly increases the odds that somebody who knows the answer will actually click on your conversation, read your question and reply. A generic scream for help only gives the impression you're too lazy to type in a clear subject, and most forum users don't like helping lazy people.

After typing in your subject and pressing Enter, the keyboard focus moves to the box empty box where you can enter the body text of your message. Please try to be as clear and descriptive as you can. The more information you provide, the more likely you'll get a timely and accurate answer. If your question is about a particular regular expression, don't forget to attach your regular expression or test data. Use the forum's attachment system rather than copying and pasting stuff into your message text.

If you want to reply to an existing conversation, select the conversation and click the Reply button. It doesn't matter which message in the conversation you selected. Replies are always to the whole conversation rather than to a particular message in a conversation. EditPad doesn't thread messages like newsgroup software tends to do. This prevents conversations from veering off-topic. If you want to respond to somebody and

bring up a different subject, you can start a new conversation, and mention the new conversation in a short reply to the old one.

When starting a reply, a new entry appears in the list of messages. Type in a summary of your reply (up to 100 characters) and press Enter. Then you can type in the full text of your reply, just like when you start a new conversation. However, doing so is optional. If your reply is very brief, simply leave the message body blank. When you send a reply without any body text, the forum system uses the summary as the body text, and automatically prepend [nt] to your summary. The [nt] is an abbreviation for "no text", meaning the summary is all there is. If you see [nt] on a reply, you don't need to click on it to see the rest of the message. This way you can quickly respond with "Thank you" or "You're welcome" and other brief courtesy messages that are often sadly absent from online communication.

When you're done with your message, click the Send button to publish it. There's no need to hurry clicking the Send button. EditPad forever keeps all your messages in progress, even if you close and restart EditPad, or refresh the forums. Sometimes it's a good idea to sleep on a reply if the discussion gets a little heated. You can have as many draft conversations and replies as you want. You can read other messages while composing your reply. If you're replying to a long question, you can switch between the message with the question and your reply while you're writing.

## Directly Attach Files and Screen Shots

One of the greatest benefits of EditPad's built-in forums is that you can attach files to your messages. Simply click the Attach button and select the item you want to attach.

To attach a screen shot, press the Print Screen button on the keyboard to capture your whole desktop. Or, press Alt+Print Screen to just capture the active window (e.g. EditPad's window). Then switch to the Forum panel, click the Attach button, and select Clipboard. You can also attach text you copied to the clipboard this way.

It's best to add your attachments while you're still composing your message. The attachments appear with the message, but won't be uploaded until you click the Send button to post your message. If you add an attachment to a message you've written previously, it is uploaded immediately. You cannot attach anything to messages written by others. Write your own reply, and attach your data to that.

To check out an attachment uploaded by somebody else, click the Use or Save button. The Use button loads the attachment directly into a new tab in EditPad, even if the attachment is not a plain text file. If you click the Save button, EditPad prompts for a location to save the attachment. EditPad does not automatically open attachments you save.

EditPad automatically compresses attachments in memory before uploading them. So if you want to attach an external file, there's no need to compress it using a zip program first. If you compress the file manually, everybody who wants to open it has to decompress it manually. If you let EditPad compress it automatically, decompression is also be automatic.

## Taking Back Your Words

If you regret anything you wrote, simply delete it. There are three Delete buttons. The one above the list of conversations deletes the whole conversation. You can only delete a conversation if nobody else participated in it. The Delete button above the edit box for the message body deletes that message, if you wrote it. The Delete button above the list of attachments deletes the selected attachment, if it belongs to a message that you wrote.

If somebody already downloaded your message before you got around to deleting it, it won't vanish magically. The message will disappear from their view of the forums the next time they log onto the forums or click Refresh. If you see messages disappear when you refresh your own view of the forums, that means the author of the message deleted it. If you replied to a conversation and the original question disappears, leaving your reply as the only message, you should delete your reply too. Otherwise, your reply will look silly all by itself. When you delete the last reply to a conversation, the conversation itself is also deleted, whether you started it or not.

## Changing Your Opinion

If you think you could better phrase something you wrote earlier, select the message and then click the Edit button above the message text. You can then edit the subject and/or body text of the message. Click the Send button to publish the edited message. It will replace the original. If you change your mind about editing the message, click the Cancel button. Make sure to click it only once! When editing a message, the Delete button changes its caption to Cancel and when clicked reverts the message to what it was before you started editing it. If you click Delete a second time (i.e. while the message is no longer being edited), you'll delete the message from the forum.

If other people have already downloaded your message, their view of the message will magically change when they click Refresh or log in again. Since things may get confusing if people respond to your original message before they see the edited message, it's best to restrict your edits to minor errors like spelling mistakes. If you change your opinion, click the Reply button to add a new message to the same conversation.

## Updating Your View

When you click the Login button, EditPad automatically downloads all new conversations and message summaries. Message bodies are downloaded one conversation at a time as you click on the conversations. Attachments are downloaded individually when you click the Use or Save button.

EditPad keeps a cache of conversations and messages that persists when you close EditPad. Attachments are cached while EditPad is running, and discarded when you close EditPad. By caching conversations and messages, EditPad improves the responsiveness of the forum while reducing the stress on the forum server.

If you keep EditPad running for a long time, EditPad does not automatically check for new conversations and messages. To do so, click the Refresh button.

Whenever you click Login or Refresh, all conversations and messages are be marked as "read". They won't have any special indicator in the list of conversations or messages. If the refresh downloads new conversation

and message summaries, those are be marked "unread". This is indicated with the same "people" icon as shown next to the Login button.

## Forum RSS Feeds

After you've logged in, you can click the Feeds button to select RSS feedsthat you can add to your favorite feed reader. This way, you can follow EditPad's discussion forums as part of your regular reading, without having to start EditPad. To participate in the discussions, simply click on a link in the RSS feed. All links in EditPad's RSS feeds will start EditPad and present the forum login screen. After you log in, wait a few moments for EditPad to download the latest conversations. EditPad will automatically select the conversation or message that the link points to. If EditPad was already running and you were already logged onto the forums, the conversation or message that the link points to is selected immediately.

You can choose which conversations should be included in the RSS feed:

- All conversations in all groups: show all conversations in all the discussion groups that you can access in EditPad.
- All conversations in the selected group: show the list of conversations that EditPad is presently showing in the Forum window.
- All conversations you participated in: show all conversations that you started or replied to in all the discussion groups that you can access in EditPad.
- All conversations you started: show all conversations that you started in all the discussion groups that you can access in EditPad.
- Only the selected conversation: show only the conversation that you are presently reading in the Forum window in EditPad.

In addition, you can choose how the conversations that you want in your RSS feed should be arranged into items or entries in the feed:

- One item per group, with a list of conversations: Entries link to groups as a whole. The entry titles show the names of the groups. The text of each entry shows a list of conversation subjects and dates. You can click the subjects to participate in the conversation in EditPad. Choose this option if you prefer to read discussions in EditPad itself (with instant access to attachments), and you only want your RSS reader to tell you if there's anything new to be read.
- One item per conversation, without messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the date the conversation was started and last replied to. If your feed has conversations from multiple groups, those will be mixed among each other.
- One item per conversation, with a list of messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the list of replies with their summaries, author names, and dates. You can click a message summary to read the message in EditPad. If your feed has conversations from multiple groups, those will be mixed among each other.
- One item per conversation, with a list of messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the list of replies, each with their full text. If your feed has conversations from multiple groups, those will be mixed among each other. This is the best option if you want to read full discussions in your RSS reader.
- One item per message with its full text: Entries link to messages (responses to conversations). The entry titles show the message summary. The entry text shows the full text of the reply, and the conversation subject that you can click on to open the conversation in EditPad. If your feed lists

multiple conversations, replies to different conversations are mixed among each other. Choose this option if you want to read full discussions in your RSS reader, but your RSS reader does not mark old entries as unread when they are updated in the RSS feed.

## Help|Support and Feedback

Before requesting technical support, please use the version history to see if you are using the latest version of EditPad. We take pride in quickly fixing bugs and resolving problems in free minor updates. If you encounter a problem with EditPad, it is quite possible that we have already released a new version that no longer has this problem.

EditPad has a built-in user forum that allows you to easily communicate with other EditPad users. If you're having a technical problem with EditPad, you're likely not the only one. The problem may have already been discussed on the forums. So search there first and you may get an immediate answer. If you don't see your issue discussed, feel free to start a new conversation in the forum. Other EditPad users will soon chime in, probably even before a Just Great Software technical support person sees it.

However, if you have purchased EditPad, you are entitled to free technical support via email. To request technical support, send an email to support@editpadlite.com if you're using EditPad Lite, or support@editpadpro.com if you're using EditPad Pro. You can expect to receive a reply by the next business day. For instant gratification, try the forums.

## Feature Requests and Other Feedback

If you have any comments about EditPad, good or bad, suggestions for improvements, please do not hesitate to send them to our technical support department. Or better yet: post them to the forum so other EditPad users can add their vote. While we cannot implement each and every user wish, we do take all feedback into account when developing new versions of our software. Customer feedback is an essential part of Just Great Software.

## Help|Create Portable Installation

The Create Portable Installation item in the Help menu makes it easy to create a portable copy of EditPad Pro on a removable disk, USB stick or flash memory card. You can run EditPad off the removable device on any computer, without leaving any traces of EditPad on that computer.

If the capacity of the device you want to install EditPad onto is limited, you can choose not to install certain parts of EditPad on the device. The only part that is not optional is the "main application" part, which is EditPad itself. You can select if you want to copy over the help file, the syntax coloring schemes, and the file navigation schemes. If you installed one or more spell check dictionaries, you can choose which ones you want to make available on the device.

Note that the Create Portable Installation command will not delete any files from the device. If you use the Create Portable Installation command a second time, and select fewer parts than the first time, EditPad will *not* remove the deselected parts from the device.

EditPad will show you the complete list of devices that Windows reports as removable devices. External hard disks often report themselves as being hard disks rather than being removable devices. If you want to install EditPad in a portable manner on such a device, turn on the option to treat all drives as removable drives. EditPad will then create a file RemovableDrive.sys on the destination drive. This file act as a token to tell EditPad not to touch the host computer.

To proceed with the installation, click on the device you want to install onto, and click the Install button. The label showing the amount of disk space needed will indicate the parts being copied to the removable drive. Once the operation is complete, the Install onto Removable Drive screen will close automatically.

EditPad will copy itself to a fixed EditPadLite7 or EditPadPro7 subfolder off the root on the device. If you like more complex folder structures, you can move the complete EditPadLite7 or EditPadPro7 folder into a different parent folder on the device.

## Creating Portable Installations using The Installer

If you have not yet installed EditPad Pro onto your hard disk, you can create a portable installation by running EditPad's installer. On the welcome screen, click the Portable Installation button. Follow the steps. This method works even if you do not have the necessary permissions to install software onto the computer you're using.

If you run the installer on 32-bit Windows the portable install will be 32-bit. It will work on both 32-bit and 64-bit Windows. If you run the installer on 64-bit Windows, then the portable install will be 64-bit. It will only work on 64-bit Windows. You can pass the `/32` command line parameter to the installer to force it to create a 32-bit installation on 64-bit Windows.

# 24. Customizing Toolbars and Menus

EditPad's main menu, all toolbars, and most context menus are fully configurable. You can move toolbars, hide them, add and remove items, and even change item captions.

EditPad's interface is modular. All functionality other than the main editor is placed on side panels that you can rearrange and even drag off as floating windows.

Use the Custom Layouts item in the View menu to save and load your customizations.

## Moving The Menu and Toolbars

Moving the menu and toolbars can be done directly by dragging them with the mouse. First, you need to make sure the toolbars aren't locked. Right-click on the main menu or any toolbar and make sure Lock Toolbars is not ticked.

To start dragging a toolbar, click on the dotted line at the left hand edge of the toolbar. While holding down the mouse button, move the mouse pointer to the location where you want to place the toolbar. The toolbar will automatically snap into place when the mouse pointer is at a position where you can dock the toolbar. If you release the mouse button, the toolbar will stay docked. If you don't release the mouse button, the toolbar will continue to be dragged. If the mouse pointer is not at a position where you can dock the toolbar, the toolbar will float on top of EditPad's window.

When a toolbar is floating, you can resize it like any window. If you double-click its caption, the toolbar will snap back to the place where it was docked last time.

You can dock the main menu and all toolbars at various positions. You can dock them at all 4 edges of EditPad's window. If side panels are visible, you can also dock them below the caption or tab of the visible side panels. If tabs or side panels are visible that take up space between the edge of EditPad's window and the main editor (the area where you edit your files), then you can also dock toolbars at any of the 4 edges of the editor that are not adjacent to the edges of EditPad's window.

You can dock multiple toolbars at the same position and stack them horizontally or vertically or both ways. Simply drag one toolbar to the same position as another toolbar. The dragged toolbar will automatically stack itself with the other one. Exactly how it stacks itself depends on whether you move the mouse pointer near the top or bottom of the other toolbar, or over the blank space at the end of the toolbar.

If you find yourself accidentally moving toolbars around, right-click on the main menu or any toolbar and select Lock Toolbars. Then you will no longer be able to move docked toolbars around. Floating toolbars can still be moved. Select Lock Toolbars again if you want to be able to move the toolbars again.

By default, the main menu and the main toolbar are docked at the top. The search toolbar is docked at the bottom, or at the top of the search panel when the search panel is visible. Most side panels have their own toolbars that are docked below the tab or caption of the panel.

## Showing and Hiding The Menu and Toolbars

To show or hide the main menu any toolbar, right-click on the main menu bar or any visible toolbar. The context menu will give you a list of all available toolbars. Click one to show or hide it. Toolbars that belong to side panels won't be available unless the side panel is visible. Use the View menu to open the side panels.

The main menu and the main toolbar cannot be both hidden. If you hide one while the other is hidden, the other one will become visible again. All other toolbars can be shown or hidden independently.

The Tools toolbar is a special toolbar. You can show or hide it and move it around, but you cannot directly customize its contents like you can with the other toolbars. The Tools toolbar lists the tools that you have selected to appear on the Tools toolbar via Tools|Configure Tools.

## Adding and Removing items from The Main Menu, Toolbars, and Context Menus

In EditPad, the main menu is really just another toolbar. The only difference is that by default it contains menu items rather than buttons. But you can place buttons directly on the main menu, and you can place menu items on toolbars and even in context menus. There is no difference between menu items and toolbar buttons. Each command can be a menu item or a toolbar button, depending on whether you place it into a menu or onto a toolbar.

To edit any toolbar or menu, right-click on any toolbar or the main menu and select Customize. The Customization screen will pop up.

To move a menu item or button from one toolbar to another, left-click on the item, hold down the mouse button, and drag the item to its new location. If you want to move an item into a submenu, move it to the menu, wait for the menu to expand while holding down the mouse button, and then move the button you're dragging to its location. To duplicate a menu item or button, hold down the Control key on the keyboard and then drag the item to its new location.

To remove a menu item or button, left-click the item and drag it to any position on the screen that is not an EditPad toolbar or menu. Then release the mouse pointer. Or, right-click the item and select Delete.

To add a separator line between menu items or toolbar buttons, first add the items that you want to separate to the menu or toolbar. Then right-click on the item that goes after the separator, and select Begin a Group.

If you want to bring back a menu item or button that you previously removed, click on the Commands tab in the Customization screen. The commands are organized into categories that correspond with the menus that contain those commands by default. The categories in the Customization screen and the items they contain never change.

Pay attention to similarly named commands in different categories. For example, you can find an Open command in the File category, the FTP Panel category, and the History Panel category. These are 3 unique commands. The File, Open command sits in the file menu and on the main toolbar by default. It shows a dialog box for opening files. The FTP Panel, Open command sits on the middle toolbar on the FTP panel. It opens the file that you have selected in the FTP panel. The File History, Open command sits on the toolbar at the top of the History Panel. It opens the file that you have selected in the History panel. You can

technically place these 3 commands anywhere. But it's probably not a good idea to use the Open items for the side panels anywhere but the proper side panel toolbar.

While customizing the toolbars and menus, an extra toolbar appears docked at the top edge of EditPad's window, below the main menu and main toolbar if they're still in their default positions. This toolbar has 6 drop-down menus (8 in EditPad Pro) that hold the items for EditPad's most important context menus. Do not customize the toolbar itself. EditPad will automatically reset it. The purpose of this toolbar is to give you access to the context menus while customizing the toolbars and menus, so you can customize the context menus too. When you're not customizing, the context menus appear in the following situations:

- Editor (outside selection): Right-click on any text in the main editor that is not selected. This menu lists commands that affect the whole file by default. If you find it confusing that you can't copy text when clicking outside the selection, add the copy commands to this menu too.
- Editor (inside selection): Right-click on selected text in the main editor. This menu lists commands for copying and editing the selected text.
- Editor (margin): Right-click on the left hand margin that shows line numbers, bookmarks, and/or folding ranges in the main editor. This menu has commands for bookmarks, folding, and line numbers.
- File tabs: Right-click on any tab for a file. This menu lists commands for saving and closing the file and setting some of its options.
- File tabs background: Right-click on the blank space after the last file tab. If there is no blank space, right-click on the scrolling arrows after the rightmost visible tab. This menu lists commands for saving and closing all files, as well as opening more files.
- Project tabs: Right-click on any tab for a project. This menu lists commands for saving and closing the project and managing the files in it.
- Project tabs background: Right-click on the blank space after the last project tab. If there is no blank space, right-click on the scrolling arrows after the rightmost visible tab. This menu lists commands for saving and closing all projects, as well as opening more projects.
- Notification icon: If you have enabled the notification icon in the System Preferences, right-clicking that icon shows this menu. It has commands for restoring and exiting EditPad, and for opening files and projects.

## Editing Menu Item and Toolbar Button Captions

Before you can edit menu items and toolbar buttons, you need to be customizing the toolbars and menus. To do so, right-click any toolbar or menu and select Customize.

To change the caption of any menu item, right-click on the menu item. In the context menu, left-click on the Name item. Then type in the new name. Press Enter to confirm the change. If you don't press Enter, the change won't stick.

To change the Alt+letter hotkey of a menu item, edit its name. In the new name, change the position of the ampersand. The letter after the last ampersand in the name becomes the Alt+letter hotkey. Use two ampersands if you want to have a literal ampersand in the name.

If you put a command that has an icon associated with it on a toolbar, the toolbar button will only show the icon by default. If you find some of the icons confusingly similar, right-click on the button and select "image and text" or "text only (always)". Then the button will show the command's caption, with or without the image. Commands that don't have an icon can also be put on toolbars. They will show their caption. The

default captions are intended for the menus. They're probably longer than you want for toolbars. You can change the caption of a button by right-clicking it, left-clicking Name, typing in the new caption, and pressing Enter.

If you want to restore the default caption of an item or button, right-click it and select Reset.

If you create multiple toolbar buttons and/or menu items for the same command, editing the caption of one of its items or buttons only affects that item or button. By default, the various search options are listed with their full captions in the Search Options submenu of the Search menu. The search toolbar has the same search options as buttons with one-word captions.

## Adding Toolbars

In the customization window, click on the Toolbars tab. This tab lists all available toolbars. Toolbars that belong to side panels won't be available unless the side panel is visible. Use the View menu to open the side panels before customizing the toolbars. Tick or clear the checkbox next to a toolbar to show or hide it.

To add a toolbar, click the New button. The name you give the toolbar will appear in the context menu when you right-click any toolbar. The new toolbar will be blank initially. You can put buttons on it like you can on any other toolbar. Adding new toolbars can be useful if you want to have toolbar buttons in different places. If you have so many buttons on the main toolbar that it wraps into multiple lines, you may prefer to move some of the buttons onto additional toolbars and dock them left or right instead of having one fat toolbar at the top.

The default toolbars can be hidden but cannot be deleted. Only toolbars you've added yourself can be deleted.

# 25. Command Line Parameters

You can specify as many files as you want on the command line. EditPad will open all files. You should put double quotes around file names that contain spaces. Putting double quotes around file names without spaces makes no difference. If you specify a file that does not exist on the command line, EditPad will create a blank tab with that file name. The file itself will not be created until you save it.

Example: `EditPadPro7.exe "C:\My Documents\text.txt" C:\Development\source.c`

If you start EditPad while another copy of EditPad is already running, the newly run copy will send the command line parameters to the existing copy. The newly run copy will close itself as soon as the existing copy as processed the command line parameters and opened all files.

If you want to force a second EditPad window to appear, specify the `/newinstance` parameter on the command line. This parameter can appear anywhere on the command line, and can be used in combination with any other command line parameter.

When using `/newinstance`, you can also specify the location of the new EditPad window. `/brl100t200r300b400` sets the window's bounding rectangle to left 100, top 200, right 300 and bottom 400, counting pixels from the top left corner of the screen.

Some applications that launch a text editor wait for the editor to close as a signal that you're done editing the file and that the application can proceed with whatever it was doing. Such applications won't behave correctly when EditPad reuses an existing instance and then closes as soon as the file has been opened. To avoid this problem, either specify the `/newinstance` parameter to start a new EditPad instance that the application can wait for. Or, specify the `/wait` parameter to reuse the existing window as usual, but to make the newly run copy wait until you have closed the file in the existing instance. While waiting, the newly run instance appears as a separate process in the task manager, but is otherwise invisible.

## Parameters Specific to One File

All of the parameters in this section require a single file to be specified on the command line. Each of them can appear only once on the command line. EditPad will not show any error messages if you do not respect these rules, but the result will not be what you expected. The order of the parameters does not matter.

`/p` tells EditPad to show the print preview immediately after opening the file, so you can print it with one click.

`/l123` (slash el one two three) tells EditPad to place the text cursor on line 123 of the file. The first line in the file is number one. The lines are counted as if word wrap were off. `/l-1` (slash el minus one) will place the text cursor at the very end of the file.

`/c45` tells EditPad to place the text cursor on character 45 of the file, counting characters from the start of the file. The first character is number zero.

If you use `/c` in combination with `/l`, then the meaning of `/c` changes. `/c7` will then place the text cursor on the 7th character on the line indicated by `/l`, counting only characters on that line, rather than counting all characters from the beginning of the file. The first character on the line is number one.

Finally, `/s123-145` will select characters 123 through 144, and place the text cursor at character 145. Like `/c`, `/s` counts characters from the start of the file, with the first character being number zero.

In combination with `/l`, `/s4-7` will select columns 4 through 6, and put the text cursor on the 7th column on the line indicated by `/l`. Again, the first character on the line is number one. `/c` and `/s` cannot be used in combination with each other.

## Additional EditPad Pro Parameters

The parameters described above are supported by both EditPad Lite and Pro. EditPad Pro supports several additional parameters.

`/newproject` tells EditPad Pro to use Project|New Project to start with a new project to open all the files specified on the command line. This parameter only makes a difference when reusing an existing instance that already has open files.

`/newprojectcombine` does the same as `/newproject`, except that `/newprojectcombine` is ignored if another file was opened with this parameter less than one second ago. If you have an application or script that launches multiple EditPad Pro instances at the same time to open multiple files at the same time, the result of the `/newprojectcombine` parameter is that all those files will be opened into one new project.

`/folder` followed by a separate parameter with the full path to a folder makes EditPad Pro show the Project|Open Folder window with that folder preselected.

`/import` shows the Project|Import File Listing screen. All the files specified on the command line are treated as file listings to be imported rather than as files to be opened.

`/hex` tells EditPad Pro to open the files specified on the command line in hexadecimal mode, regardless of whether you set this option in the file type configuration or not.

`/readonly` tells EditPad Pro to open the files specified on the command line in read-only mode. Use this parameter if you want to make sure you don't accidentally modify the files in EditPad Pro. You can click the read-only indicator on the statusbar to turn off read-only mode later.

## Part 2

# Regular Expression Tutorial

# 1. Regular Expression Tutorial

In this tutorial, I will teach you all you need to know to be able to craft powerful time-saving regular expressions. I will start with the most basic concepts, so that you can follow this tutorial even if you know nothing at all about regular expressions yet.

But I will not stop there. I will also explain how a regular expression engine works on the inside, and alert you at the consequences. This will help you to understand quickly why a particular regex does not do what you initially expected. It will save you lots of guesswork and head scratching when you need to write more complex regexes.

## What Regular Expressions Are Exactly - Terminology

Basically, a regular expression is a pattern describing a certain amount of text. Their name comes from the mathematical theory on which they are based. But we will not dig into that. Since most people including myself are lazy to type, you will usually find the name abbreviated to regex or regexp. I prefer regex, because it is easy to pronounce the plural "regexes". In this book, regular expressions are printed between guillemots: «`regex`». They clearly separate the pattern from the surrounding text and punctuation.

This first example is actually a perfectly valid regex. It is the most basic pattern, simply matching the literal text „`regex`". A "match" is the piece of text, or sequence of bytes or characters that pattern was found to correspond to by the regex processing software. Matches are indicated by double quotation marks, with the left one at the base of the line.

«`\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b`» is a more complex pattern. It describes a series of letters, digits, dots, underscores, percentage signs and hyphens, followed by an at sign, followed by another series of letters, digits and hyphens, finally followed by a single dot and between two and four letters. In other words: this pattern describes an email address.

With the above regular expression pattern, you can search through a text file to find email addresses, or verify if a given string looks like an email address. In this tutorial, I will use the term "string" to indicate the text that I am applying the regular expression to. I will indicate strings using regular double quotes. The term "string" or "character string" is used by programmers to indicate a sequence of characters. In practice, you can use regular expressions with whatever data you can access using the application or programming language you are working with.

## Different Regular Expression Engines

A regular expression "engine" is a piece of software that can process regular expressions, trying to match the pattern to the given string. Usually, the engine is part of a larger application and you do not access the engine directly. Rather, the application will invoke it for you when needed, making sure the right regular expression is applied to the right file or data.

As usual in the software world, different regular expression engines are not fully compatible with each other. It is not possible to describe every kind of engine and regular expression syntax (or "flavor") in this tutorial. I will focus on the regex flavor used by Perl 5, for the simple reason that this regex flavor is the most popular

one, and deservedly so. Many more recent regex engines are very similar, but not identical, to the one of Perl 5. Examples are the open source PCRE engine (used in many tools and languages like PHP), the .NET regular expression library, and the regular expression package included with version 1.4 and later of the Java JDK. I will point out to you whenever differences in regex flavors are important, and which features are specific to the Perl-derivatives mentioned above.

## Give Regexes a First Try

You can easily try the following yourself in a text editor that supports regular expressions, such as EditPad Pro. If you do not have such an editor, you can download the free evaluation version of EditPad Pro to try this out. EditPad Pro's regex engine is fully functional in the demo version. As a quick test, copy and paste the text of this page into EditPad Pro. Then select Search|Show Search Panel from the menu. In the search pane that appears near the bottom, type in «`regex`» in the box labeled "Search Text". Mark the "Regular expression" checkbox, and click the Find First button. This is the leftmost button on the search panel. See how EditPad Pro's regex engine finds the first match. Click the Find Next button, which sits next to the Find First button, to find further matches. When there are no further matches, the Find Next button's icon will flash briefly.

Now try to search using the regex «`reg(ular expressions?|ex(p|es)?)`» . This regex will find all names, singular and plural, I have used on this page to say "regex". If we only had plain text search, we would have needed 5 searches. With regexes, we need just one search. Regexes save you time when using a tool like EditPad Pro. Select Count Matches in the Search menu to see how many times this regular expression can match the file you have open in EditPad Pro.

If you are a programmer, your software will run faster since even a simple regex engine applying the above regex once will outperform a state of the art plain text search algorithm searching through the data five times. Regular expressions also reduce development time. With a regex engine, it takes only one line (e.g. in Perl, PHP, Java or .NET) or a couple of lines (e.g. in C using PCRE) of code to, say, check if the user's input looks like a valid email address.

# 2. Literal Characters

The most basic regular expression consists of a single literal character, e.g.: «a». It will match the first occurrence of that character in the string. If the string is "Jack is a boy", it will match the „a" after the "J". The fact that this "a" is in the middle of the word does not matter to the regex engine. If it matters to you, you will need to tell that to the regex engine by using word boundaries. We will get to that later.

This regex can match the second „a" too. It will only do so when you tell the regex engine to start searching through the string after the first match. In a text editor, you can do so by using its "Find Next" or "Search Forward" function. In a programming language, there is usually a separate function that you can call to continue searching through the string after the previous match.

Similarly, the regex «cat» will match „cat" in "About cats and dogs". This regular expression consists of a series of three literal characters. This is like saying to the regex engine: find a «c», immediately followed by an «a», immediately followed by a «t».

Note that regex engines are case sensitive by default. «cat» does not match "Cat", unless you tell the regex engine to ignore differences in case.

## Special Characters

Because we want to do more than simply search for literal pieces of text, we need to reserve certain characters for special use. In the regex flavors discussed in this tutorial, there are 11 characters with special meanings: the opening square bracket «[», the backslash «\», the caret «^», the dollar sign «$», the period or dot «.», the vertical bar or pipe symbol «|», the question mark «?», the asterisk or star «*», the plus sign «+», the opening round bracket «(» and the closing round bracket «)». These special characters are often called "metacharacters".

If you want to use any of these characters as a literal in a regex, you need to escape them with a backslash. If you want to match „1+1=2", the correct regex is «1\+1=2». Otherwise, the plus sign will have a special meaning.

Note that «1+1=2», with the backslash omitted, is a valid regex. So you will not get an error message. But it will not match "1+1=2". It would match „111=2" in "123+111=234", due to the special meaning of the plus character.

If you forget to escape a special character where its use is not allowed, such as in «+1», then you will get an error message.

Most regular expression flavors treat the brace «{» as a literal character, unless it is part of a repetition operator like «{1,3}». So you generally do not need to escape it with a backslash, though you can do so if you want. An exception to this rule is the java.util.regex package: it requires all literal braces to be escaped.

All other characters should not be escaped with a backslash. That is because the backslash is also a special character. The backslash in combination with a literal character can create a regex token with a special meaning. E.g. «\d» will match a single digit from 0 to 9.

Escaping a single metacharacter with a backslash works in all regular expression flavors. Many flavors also support the `\Q...\E` escape sequence. All the characters between the `\Q` and the `\E` are interpreted as literal characters. E.g. «`\Q*\d+*\E`» matches the literal text „`*\d+*`". The `\E` may be omitted at the end of the regex, so «`\Q*\d+*`» is the same as «`\Q*\d+*\E`». This syntax is supported by the JGsoft engine, Perl, PCRE and Java, both inside and outside character classes. However, in Java, this feature does not work correctly in JDK 1.4 and 1.5 when used in a character class or followed by a quantifier.

## Special Characters and Programming Languages

If you are a programmer, you may be surprised that characters like the single quote and double quote are not special characters. That is correct. When using a regular expression or grep tool like PowerGREP or the search function of a text editor like EditPad Pro, you should not escape or repeat the quote characters like you do in a programming language.

In your source code, you have to keep in mind which characters get special treatment inside strings by your programming language. That is because those characters will be processed by the compiler, before the regex library sees the string. So the regex «`1\+1=2`» must be written as `"1\\+1=2"` in C++ code. The C++ compiler will turn the escaped backslash in the source code into a single backslash in the string that is passed on to the regex library. To match „`c:\temp`", you need to use the regex «`c:\\temp`». As a string in C++ source code, this regex becomes `"c:\\\\temp"`. Four backslashes to match a single one indeed.

See the tools and languages section in this book for more information on how to use regular expressions in various programming languages.

## Non-Printable Characters

You can use special character sequences to put non-printable characters in your regular expression. Use «`\t`» to match a tab character (ASCII 0x09), «`\r`» for carriage return (0x0D) and «`\n`» for line feed (0x0A). More exotic non-printables are «`\a`» (bell, 0x07), «`\e`» (escape, 0x1B), «`\f`» (form feed, 0x0C) and «`\v`» (vertical tab, 0x0B). Remember that Windows text files use "`\r\n`" to terminate lines, while UNIX text files use "`\n`".

You can include any character in your regular expression if you know its hexadecimal ASCII or ANSI code for the character set that you are working with. In the Latin-1 character set, the copyright symbol is character 0xA9. So to search for the copyright symbol, you can use «`\xA9`». Another way to search for a tab is to use «`\x09`». Note that the leading zero is required.

Most regex flavors also support the tokens «`\cA`» through «`\cZ`» to insert ASCII control characters. The letter after the backslash is always a lowercase `c`. The second letter is an uppercase letter A through Z, to indicate Control+A through Control+Z. These are equivalent to «`\x01`» through «`\x1A`» (26 decimal). E.g. «`\cM`» matches a carriage return, just like «`\r`» and «`\x0D`». In XML Schema regular expressions, «`\c`» is a shorthand character class that matches any character allowed in an XML name.

If your regular expression engine supports Unicode, use «`\uFFFF`» rather than «`\xFF`» to insert a Unicode character. The euro currency sign occupies code point 0x20AC. If you cannot type it on your keyboard, you can insert it into a regular expression with «`\u20AC`».

# 3. First Look at How a Regex Engine Works Internally

Knowing how the regex engineworks will enable you to craft better regexes more easily. It will help you understand quickly why a particular regex does not do what you initially expected. This will save you lots of guesswork and head scratching when you need to write more complex regexes.

There are two kinds of regular expression engines: text-directed engines, and regex-directed engines. Jeffrey Friedl calls them DFA and NFA engines, respectively. All the regex flavors treated in this tutorial are based on regex-directed engines. This is because certain very useful features, such as lazy quantifiers and backreferences, can only be implemented in regex-directed engines. No surprise that this kind of engine is more popular.

Notable tools that use text-directed engines are awk, egrep, flex, lex, MySQL and Procmail. For awk and egrep, there are a few versions of these tools that use a regex-directed engine.

You can easily find out whether the regex flavor you intend to use has a text-directed or regex-directed engine. If backreferences and/or lazy quantifiers are available, you can be certain the engine is regex-directed. You can do the test by applying the regex «`regex|regex not`» to the string "`regex not`". If the resulting match is only „`regex`", the engine is regex-directed. If the result is „`regex not`", then it is text-directed. The reason behind this is that the regex-directed engine is "eager".

In this tutorial, after introducing a new regex token, I will explain step by step how the regex engine actually processes that token. This inside look may seem a bit long-winded at certain times. But understanding how the regex engine works will enable you to use its full power and help you avoid common mistakes.

## The Regex-Directed Engine Always Returns the Leftmost Match

This is a very important point to understand: a regex-directed engine will always return the leftmost match, even if a "better" match could be found later. When applying a regex to a string, the engine will start at the first character of the string. It will try all possible permutations of the regular expression at the first character. Only if all possibilities have been tried and found to fail, will the engine continue with the second character in the text. Again, it will try all possible permutations of the regex, in exactly the same order. The result is that the regex-directed engine will return the *leftmost* match.

When applying «`cat`» to "`He captured a catfish for his cat.`", the engine will try to match the first token in the regex «`c`» to the first character in the match "`H`". This fails. There are no other possible permutations of this regex, because it merely consists of a sequence of literal characters. So the regex engine tries to match the «`c`» with the "`e`". This fails too, as does matching the «`c`» with the space. Arriving at the 4th character in the match, «`c`» matches „`c`". The engine will then try to match the second token «`a`» to the 5th character, „`a`". This succeeds too. But then, «`t`» fails to match "`p`". At that point, the engine knows the regex cannot be matched starting at the 4th character in the match. So it will continue with the 5th: "`a`". Again, «`c`» fails to match here and the engine carries on. At the 15th character in the match, «`c`» again matches „`c`". The engine then proceeds to attempt to match the remainder of the regex at character 15 and finds that «`a`» matches „`a`" and «`t`» matches „`t`".

The entire regular expression could be matched starting at character 15. The engine is "eager" to report a match. It will therefore report the first three letters of catfish as a valid match. The engine never proceeds beyond this point to see if there are any "better" matches. The first match is considered good enough.

In this first example of the engine's internals, our regex engine simply appears to work like a regular text search routine. A text-directed engine would have returned the same result too. However, it is important that you can follow the steps the engine takes in your mind. In following examples, the way the engine works will have a profound impact on the matches it will find. Some of the results may be surprising. But they are always logical and predetermined, once you know how the engine works.

# 4. Character Classes or Character Sets

With a "character class", also called “character set”, you can tell the regex engine to match only one out of several characters. Simply place the characters you want to match between square brackets. If you want to match an a or an e, use «`[ae]`». You could use this in «`gr[ae]y`» to match either „`gray`” or „`grey`”. Very useful if you do not know whether the document you are searching through is written in American or British English.

A character class matches only a single character. «`gr[ae]y`» will not match “`graay`”, “`graey`” or any such thing. The order of the characters inside a character class does not matter. The results are identical.

You can use a hyphen inside a character class to specify a range of characters. «`[0-9]`» matches a *single* digit between 0 and 9. You can use more than one range. «`[0-9a-fA-F]`» matches a single hexadecimal digit, case insensitively. You can combine ranges and single characters. «`[0-9a-fxA-FX]`» matches a hexadecimal digit or the letter X. Again, the order of the characters and the ranges does not matter.

## Useful Applications

Find a word, even if it is misspelled, such as «`sep[ae]r[ae]te`» or «`li[cs]en[cs]e`».

Find an identifier in a programming language with «`[A-Za-z_][A-Za-z_0-9]*`».

Find a C-style hexadecimal number with «`0[xX][A-Fa-f0-9]+`».

## Negated Character Classes

Typing a caret after the opening square bracket will negate the character class. The result is that the character class will match any character that is *not* in the character class. Unlike the dot, negated character classes also match (invisible) line break characters.

It is important to remember that a negated character class still must match a character. «`q[^u]`» does *not* mean: “a q not followed by a u”. It means: “a q followed by a character that is not a u”. It will not match the q in the string “`Iraq`”. It will match the q and the space after the q in “`Iraq is a country`”. Indeed: the space will be part of the overall match, because it is the “character that is not a u” that is matched by the negated character class in the above regexp. If you want the regex to match the q, and only the q, in both strings, you need to use negative lookahead: «`q(?!u)`». But we will get to that later.

## Metacharacters Inside Character Classes

Note that the only special characters or metacharacters inside a character class are the closing bracket (]), the backslash (\), the caret (^) and the hyphen (-). The usual metacharacters are normal characters inside a character class, and do not need to be escaped by a backslash. To search for a star or plus, use «`[+*]`». Your regex will work fine if you escape the regular metacharacters inside a character class, but doing so significantly reduces readability.

To include a backslash as a character without any special meaning inside a character class, you have to escape it with another backslash. «`[\\x]`» matches a backslash or an x. The closing bracket (]), the caret (^) and the hyphen (-) can be included by escaping them with a backslash, or by placing them in a position where they do not take on their special meaning. I recommend the latter method, since it improves readability. To include a caret, place it anywhere except right after the opening bracket. «`[x^]`» matches an x or a caret. You can put the closing bracket right after the opening bracket, or the negating caret. «`[]x]`» matches a closing bracket or an x. «`[^]x]`» matches any character that is not a closing bracket or an x. The hyphen can be included right after the opening bracket, or right before the closing bracket, or right after the negating caret. Both «`[-x]`» and «`[x-]`» match an x or a hyphen.

You can use all non-printable characters in character classes just like you can use them outside of character classes. E.g. «`[$\u20AC]`» matches a dollar or euro sign, assuming your regex flavor supports Unicode.

The JGsoft engine, Perl and PCRE also support the \Q...\E sequence inside character classes to escape a string of characters. E.g. «`[\Q[-]\E]`» matches „[", „-" or „]".

POSIX regular expressions treat the backslash as a literal character inside character classes. This means you can't use backslashes to escape the closing bracket (]), the caret (^) and the hyphen (-). To use these characters, position them as explained above in this section. This also means that special tokens like shorthands are not available in POSIX regular expressions. See the tutorial topic on POSIX bracket expressions for more information.

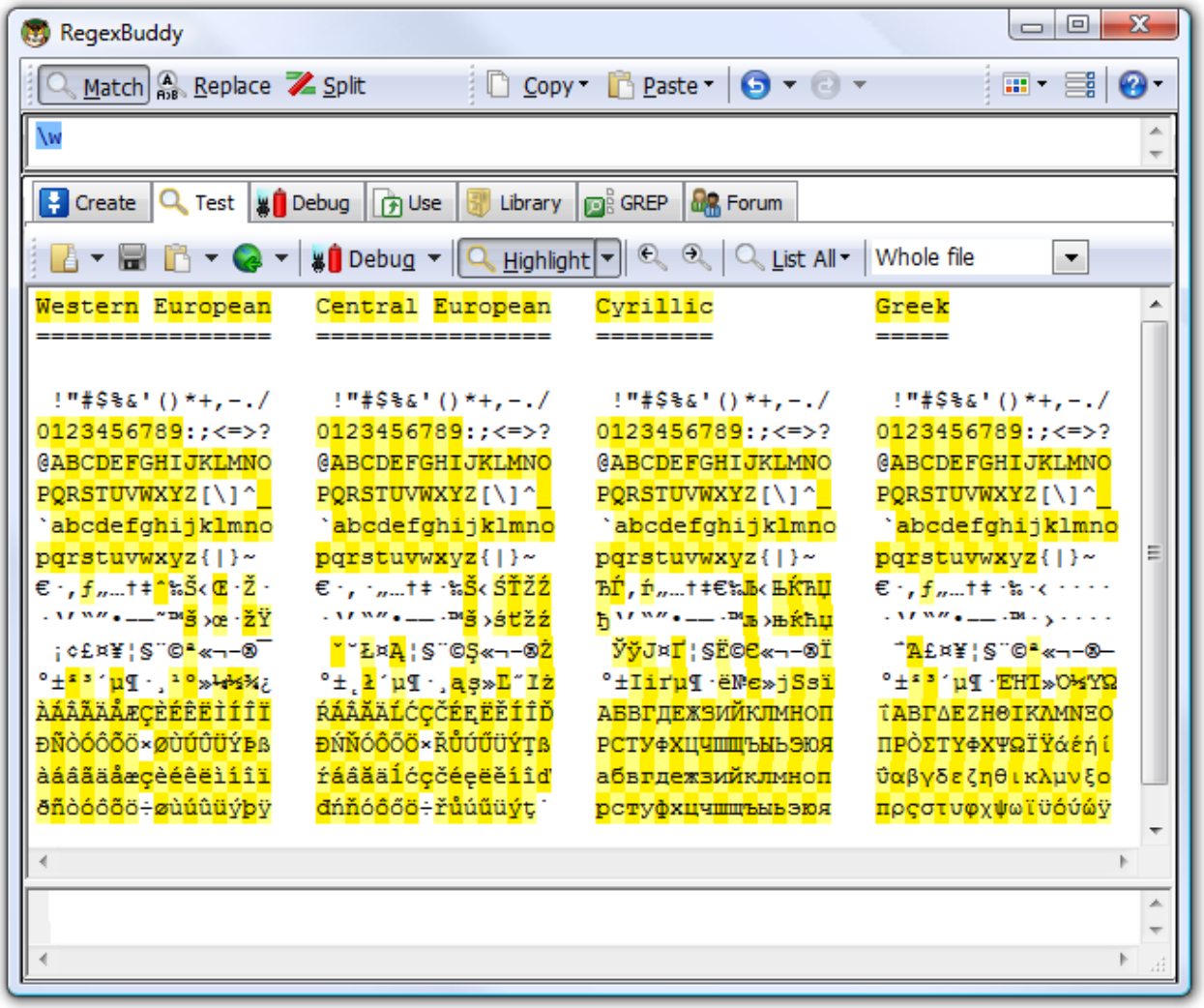
## Shorthand Character Classes

Since certain character classes are used often, a series of shorthand character classes are available. «`\d`» is short for «`[0-9]`».

«`\w`» stands for "word character", usually «`[A-Za-z0-9_]`». Notice the inclusion of the underscore and digits.

«`\s`» stands for "whitespace character". Again, which characters this actually includes, depends on the regex flavor. In all flavors discussed in this tutorial, it includes «`[ \t\r\n]`». That is: «`\s`» will match a space, a tab or a line break. Some flavors include additional, rarely used non-printable characters such as vertical tab and form feed.

The flavor comparison shows "ascii only" for flavors that match only the ASCII characters listed in the previous paragraphs. With flavors marked as "YES", letters, digits and space characters from other languages or Unicode are also included in the shorthand classes. In the screen shot, you can see the characters matched by «`\w`» in RegexBuddy using various scripts. Notice that JavaScript uses ASCII for «`\d`» and «`\w`», but Unicode for «`\s`». XML does it the other way around. Python offers flags to control what the shorthands should match.




Shorthand character classes can be used both inside and outside the square brackets. «`\s\d`» matches a whitespace character followed by a digit. «`[\s\d]`» matches a single character that is either whitespace or a digit. When applied to "`1 + 2 = 3`", the former regex will match „ `2`" (space two), while the latter matches „`1`" (one). «`[\da-fA-F]`» matches a hexadecimal digit, and is equivalent to «`[0-9a-fA-F]`».

## Negated Shorthand Character Classes

The above three shorthands also have negated versions. «`\D`» is the same as «`[^\d]`», «`\W`» is short for «`[^\w]`» and «`\S`» is the equivalent of «`[^\s]`».

Be careful when using the negated shorthands inside square brackets. «`[\D\S]`» is *not* the same as «`[^\d\s]`». The latter will match any character that is not a digit or whitespace. So it will match „`x`", but not "`8`". The former, however, will match any character that is either not a digit, or is not whitespace. Because a digit is not whitespace, and whitespace is not a digit, «`[\D\S]`» will match any character, digit, whitespace or otherwise.

## Repeating Character Classes

If you repeat a character class by using the «?», «*» or «+» operators, you will repeat the entire character class, and not just the character that it matched. The regex «`[0-9]+`» can match „`837`" as well as „`222`".

If you want to repeat the matched character, rather than the class, you will need to use backreferences. «`([0-9])\1+`» will match „`222`" but not "`837`". When applied to the string "`833337`", it will match „`3333`" in the middle of this string. If you do not want that, you need to use lookahead and lookbehind.

But I digress. I did not yet explain how character classes work inside the regex engine. Let us take a look at that first.

## Looking Inside The Regex Engine

As I already said: the order of the characters inside a character class does not matter. «`gr[ae]y`» will match „`grey`" in "`Is his hair grey or gray?`", because that is the *leftmost match*. We already saw how the engine applies a regex consisting only of literal characters. Below, I will explain how it applies a regex that has more than one permutation. That is: «`gr[ae]y`» can match both „`gray`" and „`grey`".

Nothing noteworthy happens for the first twelve characters in the string. The engine will fail to match «`g`» at every step, and continue with the next character in the string. When the engine arrives at the 13th character, „`g`" is matched. The engine will then try to match the remainder of the regex with the text. The next token in the regex is the literal «`r`», which matches the next character in the text. So the third token, «`[ae]`» is attempted at the next character in the text ("`e`"). The character class gives the engine two options: match «`a`» or match «`e`». It will first attempt to match «`a`», and fail.

But because we are using a regex-directed engine, it must continue trying to match all the other permutations of the regex pattern before deciding that the regex cannot be matched with the text starting at character 13. So it will continue with the other option, and find that «`e`» matches „`e`". The last regex token is «`y`», which can be matched with the following character as well. The engine has found a complete match with the text starting at character 13. It will return „`grey`" as the match result, and look no further. Again, the *leftmost match* was returned, even though we put the «`a`» first in the character class, and „`gray`" could have been matched in the string. But the engine simply did not get that far, because another equally valid match was found to the left of it.

# 5. The Dot Matches (Almost) Any Character

In regular expressions, the dot or period is one of the most commonly used metacharacters. Unfortunately, it is also the most commonly misused metacharacter.

The dot matches a single character, without caring what that character is. The only exception are newlinecharacters. In all regex flavors discussed in this tutorial, the dot will *not* match a newline character by default. So by default, the dot is short for the negated character class «`[^\n]`» (UNIX regex flavors) or «`[^\r\n]`» (Windows regex flavors).

This exception exists mostly because of historic reasons. The first tools that used regular expressions were line-based. They would read a file line by line, and apply the regular expression separately to each line. The effect is that with these tools, the string could never contain newlines, so the dot could never match them.

Modern tools and languages can apply regular expressions to very large strings or even entire files. All regex flavors discussed here have an option to make the dot match all characters, including newlines. In RegexBuddy, EditPad Pro or PowerGREP, you simply tick the checkbox labeled "dot matches newline".

In Perl, the mode where the dot also matches newlines is called "single-line mode". This is a bit unfortunate, because it is easy to mix up this term with "multi-line mode". Multi-line mode only affects anchors, and single-line mode only affects the dot. You can activate single-line mode by adding an s after the regex code, like this: `m/^regex$/s;`.

Other languages and regex libraries have adopted Perl's terminology. When using the regex classes of the .NET framework, you activate this mode by specifying `RegexOptions.Singleline`, such as in `Regex.Match("string", "regex", RegexOptions.Singleline)`.

In all programming languages and regex libraries I know, activating single-line mode has no effect other than making the dot match newlines. So if you expose this option to your users, please give it a clearer label like was done in RegexBuddy, EditPad Pro and PowerGREP.

JavaScript and VBScript do not have an option to make the dot match line break characters. In those languages, you can use a character class such as «`[\s\S]`» to match any character. This character matches a character that is either a whitespace character (including line break characters), or a character that is not a whitespace character. Since all characters are either whitespace or non-whitespace, this character class matches any character.

## Use The Dot Sparingly

The dot is a very powerful regex metacharacter. It allows you to be lazy. Put in a dot, and everything will match just fine when you test the regex on valid data. The problem is that the regex will also match in cases where it should not match. If you are new to regular expressions, some of these cases may not be so obvious at first.

I will illustrate this with a simple example. Let's say we want to match a date in mm/dd/yy format, but we want to leave the user the choice of date separators. The quick solution is «`\d\d.\d\d.\d\d`». Seems fine at first. It will match a date like „`02/12/03`" just fine. Trouble is: „`02512703`" is also considered a valid date by

this regular expression. In this match, the first dot matched „5", and the second matched „7". Obviously not what we intended.

«\d\d[- /.]\d\d[- /.]\d\d» is a better solution. This regex allows a dash, space, dot and forward slash as date separators. Remember that the dot is not a metacharacter inside a character class, so we do not need to escape it with a backslash.

This regex is still far from perfect. It matches „99/99/99" as a valid date. «[0-1]\d[- /.][0-3]\d[- /.]\d\d» is a step ahead, though it will still match „19/39/99". How perfect you want your regex to be depends on what you want to do with it. If you are validating user input, it has to be perfect. If you are parsing data files from a known source that generates its files in the same way every time, our last attempt is probably more than sufficient to parse the data without errors. You can find a better regex to match dates in the example section.

## Use Negated Character Sets Instead of the Dot

I will explain this in depth when I present you the repeat operators star and plus, but the warning is important enough to mention it here as well. I will illustrate with an example.

Suppose you want to match a double-quoted string. Sounds easy. We can have any number of any character between the double quotes, so «".*"» seems to do the trick just fine. The dot matches any character, and the star allows the dot to be repeated any number of times, including zero. If you test this regex on "Put a "string" between double quotes", it will match „"string"" just fine. Now go ahead and test it on "Houston, we have a problem with "string one" and "string two". Please respond."

Ouch. The regex matches „"string one" and "string two"". Definitely not what we intended. The reason for this is that the star is *greedy*.

In the date-matching example, we improved our regex by replacing the dot with a character class. Here, we will do the same. Our original definition of a double-quoted string was faulty. We do not want any number of *any character* between the quotes. We want any number of characters that are not double quotes or newlines between the quotes. So the proper regex is «"[^"\r\n]*"».

# 6. Start of String and End of String Anchors

Thus far, I have explained literal characters and character classes. In both cases, putting one in a regex will cause the regex engine to try to match a single character.

Anchors are a different breed. They do not match any character at all. Instead, they match a position before, after or between characters. They can be used to "anchor" the regex match at a certain position. The caret «^» matches the position before the first character in the string. Applying «`^a`» to "`abc`" matches „a". «`^b`» will not match "`abc`" at all, because the «`b`» cannot be matched right after the start of the string, matched by «^». See below for the inside view of the regex engine.

Similarly, «`$`» matches right after the last character in the string. «`c$`» matches „c" in "`abc`", while «`a$`» does not match at all.

## Useful Applications

When using regular expressions in a programming language to validate user input, using anchors is very important. If you use the code `if ($input =~ m/\d+/)` in a Perl script to see if the user entered an integer number, it will accept the input even if the user entered "`qsdf4ghjk`", because «`\d+`» matches the 4. The correct regex to use is «`^\d+$`». Because "start of string" must be matched before the match of «`\d+`», and "end of string" must be matched right after it, the entire string must consist of digits for «`^\d+$`» to be able to match.

It is easy for the user to accidentally type in a space. When Perl reads from a line from a text file, the line break will also be stored in the variable. So before validating input, it is good practice to trim leading and trailing whitespace. «`^\s+`» matches leading whitespace and «`\s+$`» matches trailing whitespace. In Perl, you could use `$input =~ s/^\s+|\s+$//g`. Handy use of alternation and /g allows us to do this in a single line of code.

## Using ^ and $ as Start of Line and End of Line Anchors

If you have a string consisting of multiple lines, like "`first line\nsecond line`" (where \n indicates a line break), it is often desirable to work with lines, rather than the entire string. Therefore, all the regex engines discussed in this tutorial have the option to expand the meaning of both anchors. «^» can then match at the start of the string (before the "`f`" in the above string), as well as after each line break (between "`\n`" and "`s`"). Likewise, «`$`» will still match at the end of the string (after the last "`e`"), and also before every line break (between "`e`" and "`\n`").

In text editors like EditPad Pro or GNU Emacs, and regex tools like PowerGREP, the caret and dollar always match at the start and end of each line. This makes sense because those applications are designed to work with entire files, rather than short strings.

In all programming languages and libraries discussed in this book , except Ruby, you have to explicitly activate this extended functionality. It is traditionally called "multi-line mode". In Perl, you do this by adding an m after the regex code, like this: `m/^regex$/m;`. In .NET, the anchors match before and after newlines when you specify `RegexOptions.Multiline`, such as in `Regex.Match("string", "regex", RegexOptions.Multiline)`.

## Permanent Start of String and End of String Anchors

«\A» only ever matches at the start of the string. Likewise, «\Z» only ever matches at the end of the string. These two tokens never match at line breaks. This is true in all regex flavors discussed in this tutorial, even when you turn on "multiline mode". In EditPad Pro and PowerGREP, where the caret and dollar always match at the start and end of lines, «\A» and «\Z» only match at the start and the end of the entire file.

JavaScript, POSIX and XML do not support «\A» and «\Z». You're stuck with using the caret and dollar for this purpose.

The GNU extensions to POSIX regular expressions use «\`» (backtick) to match the start of the string, and «\'» (single quote) to match the end of the string.

## Zero-Length Matches

We saw that the anchors match at a position, rather than matching a character. This means that when a regex only consists of one or more anchors, it can result in a zero-length match. Depending on the situation, this can be very useful or undesirable. Using «^\d*$» to test if the user entered a number (notice the use of the star instead of the plus), would cause the script to accept an empty string as a valid input. See below.

However, matching only a position can be very useful. In email, for example, it is common to prepend a "greater than" symbol and a space to each line of the quoted message. In VB.NET, we can easily do this with `Dim Quoted as String = Regex.Replace(Original, "^", "> ", RegexOptions.Multiline)`. We are using multi-line mode, so the regex «^» matches at the start of the quoted message, and after each newline. The Regex.Replace method will remove the regex match from the string, and insert the replacement string (greater than symbol and a space). Since the match does not include any characters, nothing is deleted. However, the match does include a starting position, and the replacement string is inserted there, just like we want it.

## Strings Ending with a Line Break

Even though «\Z» and «$» only match at the end of the string (when the option for the caret and dollar to match at embedded line breaks is off), there is one exception. If the string ends with a line break, then «\Z» and «$» will match at the position before that line break, rather than at the very end of the string. This "enhancement" was introduced by Perl, and is copied by many regex flavors, including Java, .NET and PCRE. In Perl, when reading a line from a file, the resulting string will end with a line break. Reading a line from a file with the text "joe" results in the string "joe\n". When applied to this string, both «^[a-z]+$» and «\A[a-z]+\Z» will match „joe".

If you only want a match at the absolute very end of the string, use «\z» (lower case z instead of upper case Z). «\A[a-z]+\z» does not match "joe\n". «\z» matches after the line break, which is not matched by the character class.

## Looking Inside the Regex Engine

Let's see what happens when we try to match «^4$» to "749\n486\n4" (where \n represents a newline character) in multi-line mode. As usual, the regex engine starts at the first character: "7". The first token in the regular expression is «^». Since this token is a zero-width token, the engine does not try to match it with the character, but rather with the position before the character that the regex engine has reached so far. «^» indeed matches the position before "7". The engine then advances to the next regex token: «4». Since the previous token was zero-width, the regex engine does *not* advance to the next character in the string. It remains at "7". «4» is a literal character, which does not match "7". There are no other permutations of the regex, so the engine starts again with the first regex token, at the next character: "4". This time, «^» cannot match at the position before the 4. This position is preceded by a character, and that character is not a newline. The engine continues at "9", and fails again. The next attempt, at "\n", also fails. Again, the position before "\n" is preceded by a character, "9", and that character is not a newline.

Then, the regex engine arrives at the second "4" in the string. The «^» can match at the position before the "4", because it is preceded by a newline character. Again, the regex engine advances to the next regex token, «4», but does not advance the character position in the string. «4» matches „4", and the engine advances both the regex token and the string character. Now the engine attempts to match «$» at the position before (indeed: before) the "8". The dollar cannot match here, because this position is followed by a character, and that character is not a newline.

Yet again, the engine must try to match the first token again. Previously, it was successfully matched at the second "4", so the engine continues at the next character, "8", where the caret does not match. Same at the six and the newline.

Finally, the regex engine tries to match the first token at the third "4" in the string. With success. After that, the engine successfully matches «4» with „4". The current regex token is advanced to «$», and the current character is advanced to the very last position in the string: the void after the string. No regex token that needs a character to match can match here. Not even a negated character class. However, we are trying to match a dollar sign, and the mighty dollar is a strange beast. It is zero-width, so it will try to match the position before the current character. It does not matter that this "character" is the void after the string. In fact, the dollar will check the current character. It must be either a newline, or the void after the string, for «$» to match the position before the current character. Since that is the case after the example, the dollar matches successfully.

Since «$» was the last token in the regex, the engine has found a successful match: the last „4" in the string.

## Another Inside Look

Earlier I mentioned that «^\d*$» would successfully match an empty string. Let's see why.

There is only one "character" position in an empty string: the void after the string. The first token in the regex is «^». It matches the position before the void after the string, because it is preceded by the void before the string. The next token is «\d*». As we will see later, one of the star's effects is that it makes the «\d», in this case, optional. The engine will try to match «\d» with the void after the string. That fails, but the star turns the failure of the «\d» into a zero-width success. The engine will proceed with the next regex token, without advancing the position in the string. So the engine arrives at «$», and the void after the string. We already saw that those match. At this point, the entire regex has matched the empty string, and the engine reports success.

## Caution for Programmers

A regular expression such as «$» all by itself can indeed match after the string. If you would query the engine for the character position, it would return the length of the string if string indices are zero-based, or the length+1 if string indices are one-based in your programming language. If you would query the engine for the length of the match, it would return zero.

What you have to watch out for is that String[Regex.MatchPosition] may cause an access violation or segmentation fault, because MatchPosition can point to the void after the string. This can also happen with «^» and «^$» if the last character in the string is a newline.

# 7. Word Boundaries

The metacharacter «\b» is an anchor like the caret and the dollar sign. It matches at a position that is called a "word boundary". This match is zero-length.

There are three different positions that qualify as word boundaries:

- Before the first character in the string, if the first character is a word character.
- After the last character in the string, if the last character is a word character.
- Between two characters in the string, where one is a word character and the other is not a word character.

Simply put: «\b» allows you to perform a "whole words only" search using a regular expression in the form of «\bword\b». A "word character" is a character that can be used to form words. All characters that are not "word characters" are "non-word characters".

In all flavors, the characters «[a-zA-Z0-9_]» are word characters. These are also matched by the short-hand character class «\w». Flavors showing "ascii" for word boundaries in the flavor comparison recognize only these as word characters. Flavors showing "YES" also recognize letters and digits from other languages or all of Unicode as word characters. Notice that Java supports Unicode for «\b» but not for «\w». Python offers flags to control which characters are word characters (affecting both «\b» and «\w»).

In Perl and the other regex flavors discussed in this tutorial, there is only one metacharacter that matches both before a word and after a word. This is because any position between characters can never be both at the start and at the end of a word. Using only one operator makes things easier for you.

Since digits are considered to be word characters, «\b4\b» can be used to match a 4 that is not part of a larger number. This regex will not match "44 sheets of a4". So saying "«\b» matches before and after an alphanumeric sequence" is more exact than saying "before and after a word".

## Negated Word Boundary

«\B» is the negated version of «\b». «\B» matches at every position where «\b» does not. Effectively, «\B» matches at any position between two word characters as well as at any position between two non-word characters.

## Looking Inside the Regex Engine

Let's see what happens when we apply the regex «\bis\b» to the string "This island is beautiful". The engine starts with the first token «\b» at the first character "T". Since this token is zero-length, the position before the character is inspected. «\b» matches here, because the T is a word character and the character before it is the void before the start of the string. The engine continues with the next token: the literal «i». The engine does not advance to the next character in the string, because the previous regex token was zero-width. «i» does not match "T", so the engine retries the first token at the next character position.

«\b» cannot match at the position between the "T" and the "h". It cannot match between the "h" and the "i" either, and neither between the "i" and the "s".

The next character in the string is a space. «\b» matches here because the space is not a word character, and the preceding character is. Again, the engine continues with the «i» which does not match with the space.

Advancing a character and restarting with the first regex token, «\b» matches between the space and the second "i" in the string. Continuing, the regex engine finds that «i» matches „i" and «s» matches „s". Now, the engine tries to match the second «\b» at the position before the "l". This fails because this position is between two word characters. The engine reverts to the start of the regex and advances one character to the "s" in "island". Again, the «\b» fails to match and continues to do so until the second space is reached. It matches there, but matching the «i» fails.

But «\b» matches at the position before the third "i" in the string. The engine continues, and finds that «i» matches „i" and «s» matches „s". The last token in the regex, «\b», also matches at the position before the third space in the string because the space is not a word character, and the character before it is.

The engine has successfully matched the word „is" in our string, skipping the two earlier occurrences of the characters i and s. If we had used the regular expression «is», it would have matched the „is" in "This".

## Tcl Word Boundaries

Word boundaries, as described above, are supported by most regular expression flavors. Notable exceptions are the POSIX and XML Schema flavors, which don't support word boundaries at all. Tcl uses a different syntax.

In Tcl, «\b» matches a backspace character, just like «\x08» in most regex flavors (including Tcl's). «\B» matches a single backslash character in Tcl, just like «\\» in all other regex flavors (and Tcl too).

Tcl uses the letter "y" instead of the letter "b" to match word boundaries. «\y» matches at any word boundary position, while «\Y» matches at any position that is not a word boundary. These Tcl regex tokens match exactly the same as «\b» and «\B» in Perl-style regex flavors. They don't discriminate between the start and the end of a word.

Tcl has two more word boundary tokens that do discriminate between the start and end of a word. «\m» matches only at the start of a word. That is, it matches at any position that has a non-word character to the left of it, and a word character to the right of it. It also matches at the start of the string if the first character in the string is a word character. «\M» matches only at the end of a word. It matches at any position that has a word character to the left of it, and a non-word character to the right of it. It also matches at the end of the string if the last character in the string is a word character.

The only regex engine that supports Tcl-style word boundaries (besides Tcl itself) is the JGsoft engine. In PowerGREP and EditPad Pro, «\b» and «\B» are Perl-style word boundaries, and «\y», «\Y», «\m» and «\M» are Tcl-style word boundaries.

In most situations, the lack of «\m» and «\M» tokens is not a problem. «\yword\y» finds "whole words only" occurrences of "word" just like «\mword\M» would. «\Mword\m» could never match anywhere, since «\M» never matches at a position followed by a word character, and «\m» never at a position preceded by one. If your regular expression needs to match characters before or after «\y», you can easily specify in the regex

whether these characters should be word characters or non-word characters. E.g. if you want to match any word, «`\y\w+\y`» will give the same result as «`\m.+\M`». Using «`\w`» instead of the dot automatically restricts the first «`\y`» to the start of a word, and the second «`\y`» to the end of a word. Note that «`\y.+\y`» would not work. This regex matches each word, and also each sequence of non-word characters between the words in your subject string. That said, if your flavor supports «`\m`» and «`\M`», the regex engine could apply «`\m\w+\M`» slightly faster than «`\y\w+\y`», depending on its internal optimizations.

If your regex flavor supports lookahead and lookbehind, you can use «`(?<!\w)(?=\w)`» to emulate Tcl's «`\m`» and «`(?<=\w)(?!\w)`» to emulate «`\M`». Though quite a bit more verbose, these lookaround constructs match exactly the same as Tcl's word boundaries.

If your flavor has lookahead but not lookbehind, and also has Perl-style word boundaries, you can use «`\b(?=\w)`» to emulate Tcl's «`\m`» and «`\b(?!\w)`» to emulate «`\M`». «`\b`» matches at the start or end of a word, and the lookahead checks if the next character is part of a word or not. If it is we're at the start of a word. Otherwise, we're at the end of a word.

## GNU Word Boundaries

The GNU extensions to POSIX regular expressions add support for the «`\b`» and «`\B`» word boundaries, as described above. GNU also uses its own syntax for start-of-word and end-of-word boundaries. «`\<`» matches at the start of a word, like Tcl's «`\m`». «`\>`» matches at the end of a word, like Tcl's «`\M`».

# 8. Alternation with The Vertical Bar or Pipe Symbol

I already explained how you can use character classes to match a single character out of several possible characters. Alternation is similar. You can use alternation to match a single regular expression out of several possible regular expressions.

If you want to search for the literal text «`cat`» or «`dog`», separate both options with a vertical bar or pipe symbol: «`cat|dog`». If you want more options, simply expand the list: «`cat|dog|mouse|fish`» .

The alternation operator has the lowest precedence of all regex operators. That is, it tells the regex engine to match either everything to the left of the vertical bar, or everything to the right of the vertical bar. If you want to limit the reach of the alternation, you will need to use round brackets for grouping. If we want to improve the first example to match whole words only, we would need to use «`\b(cat|dog)\b`». This tells the regex engine to find a word boundary, then either "cat" or "dog", and then another word boundary. If we had omitted the round brackets, the regex engine would have searched for "a word boundary followed by cat", or, "dog" followed by a word boundary.

## Remember That The Regex Engine Is Eager

I already explained that the regex engine is eager. It will stop searching as soon as it finds a valid match. The consequence is that in certain situations, the order of the alternatives matters. Suppose you want to use a regex to match a list of function names in a programming language: Get, GetValue, Set or SetValue. The obvious solution is «`Get|GetValue|Set|SetValue`». Let's see how this works out when the string is "`SetValue`".

The regex engine starts at the first token in the regex, «`G`», and at the first character in the string, "`S`". The match fails. However, the regex engine studied the entire regular expression before starting. So it knows that this regular expression uses alternation, and that the entire regex has not failed yet. So it continues with the second option, being the second «`G`» in the regex. The match fails again. The next token is the first «`S`» in the regex. The match succeeds, and the engine continues with the next character in the string, as well as the next token in the regex. The next token in the regex is the «`e`» after the «`S`» that just successfully matched. «`e`» matches „`e`". The next token, «`t`» matches „`t`".

At this point, the third option in the alternation has been successfully matched. Because the regex engine is eager, it considers the entire alternation to have been successfully matched as soon as one of the options has. In this example, there are no other tokens in the regex outside the alternation, so the entire regex has successfully matched „`Set`" in "`SetValue`".

Contrary to what we intended, the regex did not match the entire string. There are several solutions. One option is to take into account that the regex engine is eager, and change the order of the options. If we use «`GetValue|Get|SetValue|Set`», «`SetValue`» will be attempted before «`Set`», and the engine will match the entire string. We could also combine the four options into two and use the question mark to make part of them optional: «`Get(Value)?|Set(Value)?`». Because the question mark is greedy, «`SetValue`» will be attempted before «`Set`».

The best option is probably to express the fact that we only want to match complete words. We do not want to match Set or SetValue if the string is "`SetValueFunction`". So the solution is

«`\b(Get|GetValue|Set|SetValue)\b`» or «`\b(Get(Value)?|Set(Value)?)\b`». Since all options have the same end, we can optimize this further to «`\b(Get|Set)(Value)?\b`» .

All regex flavors discussed in this book work this way, except one: the POSIX standard mandates that the longest match be returned, regardless if the regex engine is implemented using an NFA or DFA algorithm.

# 9. Optional Items

The question mark makes the preceding token in the regular expression optional. E.g.: «`colou?r`» matches both „`colour`" and „`color`".

You can make several tokens optional by grouping them together using round brackets, and placing the question mark after the closing bracket. E.g.: «`Nov(ember)?`» will match „`Nov`" and „`November`".

You can write a regular expression that matches many alternatives by including more than one question mark. «`Feb(ruary)? 23(rd)?`» matches „`February 23rd`", „`February 23`", „`Feb 23rd`" and „`Feb 23`".

## Important Regex Concept: Greediness

With the question mark, I have introduced the first metacharacter that is *greedy*. The question mark gives the regex engine two choices: try to match the part the question mark applies to, or do not try to match it. The engine will always try to match that part. Only if this causes the entire regular expression to fail, will the engine try ignoring the part the question mark applies to.

The effect is that if you apply the regex «`Feb 23(rd)?`» to the string "`Today is Feb 23rd, 2003`", the match will always be „`Feb 23rd`" and not „`Feb 23`". You can make the question mark *lazy* (i.e. turn off the greediness) by putting a second question mark after the first.

I will say a lot more about greediness when discussing the other repetition operators.

## Looking Inside The Regex Engine

Let's apply the regular expression «`colou?r`» to the string "`The colonel likes the color green`".

The first token in the regex is the literal «`c`». The first position where it matches successfully is the „`c`" in "`colonel`". The engine continues, and finds that «`o`» matches „`o`", «`l`» matches „`l`" and another «`o`» matches „`o`". Then the engine checks whether «`u`» matches "`n`". This fails. However, the question mark tells the regex engine that failing to match «`u`» is acceptable. Therefore, the engine will skip ahead to the next regex token: «`r`». But this fails to match "`n`" as well. Now, the engine can only conclude that the entire regular expression cannot be matched starting at the „`c`" in "`colonel`". Therefore, the engine starts again trying to match «`c`» to the first o in "`colonel`".

After a series of failures, «`c`» will match with the „`c`" in "`color`", and «`o`», «`l`» and «`o`» match the following characters. Now the engine checks whether «`u`» matches "`r`". This fails. Again: no problem. The question mark allows the engine to continue with «`r`». This matches „`r`" and the engine reports that the regex successfully matched „`color`" in our string.

# 10. Repetition with Star and Plus

I already introduced one repetition operator or quantifier: the question mark. It tells the engine to attempt to match the preceding token zero times or once, in effect making it optional.

The asterisk or star tells the engine to attempt to match the preceding token zero or more times. The plus tells the engine to attempt to match the preceding token once or more. «`<[A-Za-z][A-Za-z0-9]*>`» matches an HTML tag without any attributes. The sharp brackets are literals. The first character class matches a letter. The second character class matches a letter or digit. The star repeats the second character class. Because we used the star, it's OK if the second character class matches nothing. So our regex will match a tag like „`<B>`". When matching „`<HTML>`", the first character class will match „`H`". The star will cause the second character class to be repeated three times, matching „`T`", „`M`" and „`L`" with each step.

I could also have used «`<[A-Za-z0-9]+>`». I did not, because this regex would match „`<1>`", which is not a valid HTML tag. But this regex may be sufficient if you know the string you are searching through does not contain any such invalid tags.

## Limiting Repetition

Modern regex flavors, like those discussed in this tutorial, have an additional repetition operator that allows you to specify how many times a token can be repeated. The syntax is `{`*`min`*`,`*`max`*`}`, where *min* is a positive integer number indicating the minimum number of matches, and *max* is an integer equal to or greater than *min* indicating the maximum number of matches. If the comma is present but *max* is omitted, the maximum number of matches is infinite. So «`{0,}`» is the same as «`*`», and «`{1,}`» is the same as «`+`». Omitting both the comma and *max* tells the engine to repeat the token exactly *min* times.

You could use «`\b[1-9][0-9]{3}\b`» to match a number between 1000 and 9999. «`\b[1-9][0-9]{2,4}\b`» matches a number between 100 and 99999. Notice the use of the word boundaries.

## Watch Out for The Greediness!

Suppose you want to use a regex to match an HTML tag. You know that the input will be a valid HTML file, so the regular expression does not need to exclude any invalid use of sharp brackets. If it sits between sharp brackets, it is an HTML tag.

Most people new to regular expressions will attempt to use «`<.+>`». They will be surprised when they test it on a string like "`This is a <EM>first</EM> test`". You might expect the regex to match „`<EM>`" and when continuing after that match, „`</EM>`".

But it does not. The regex will match „`<EM>first</EM>`". Obviously not what we wanted. The reason is that the plus is *greedy*. That is, the plus causes the regex engine to repeat the preceding token as often as possible. Only if that causes the entire regex to fail, will the regex engine *backtrack*. That is, it will go back to the plus, make it give up the last iteration, and proceed with the remainder of the regex. Let's take a look inside the regex engine to see in detail how this works and why this causes our regex to fail. After that, I will present you with two possible solutions.

Like the plus, the star and the repetition using curly braces are greedy.

## Looking Inside The Regex Engine

The first token in the regex is «<». This is a literal. As we already know, the first place where it will match is the first „<" in the string. The next token is the dot, which matches any character except newlines. The dot is repeated by the plus. The plus is *greedy*. Therefore, the engine will repeat the dot as many times as it can. The dot matches „E", so the regex continues to try to match the dot with the next character. „M" is matched, and the dot is repeated once more. The next character is the ">". You should see the problem by now. The dot matches the „>", and the engine continues repeating the dot. The dot will match all remaining characters in the string. The dot fails when the engine has reached the void after the end of the string. Only at this point does the regex engine continue with the next token: «>».

So far, «<.+» has matched „<EM>first</EM> test" and the engine has arrived at the end of the string. «>» cannot match here. The engine remembers that the plus has repeated the dot more often than is required. (Remember that the plus *requires* the dot to match only once.) Rather than admitting failure, the engine will *backtrack*. It will reduce the repetition of the plus by one, and then continue trying the remainder of the regex.

So the match of «.+» is reduced to „EM>first</EM> tes". The next token in the regex is still «>». But now the next character in the string is the last "t". Again, these cannot match, causing the engine to backtrack further. The total match so far is reduced to „<EM>first</EM> te". But «>» still cannot match. So the engine continues backtracking until the match of «.+» is reduced to „EM>first</EM". Now, «>» can match the next character in the string. The last token in the regex has been matched. The engine reports that „<EM>first</EM>" has been successfully matched.

Remember that the regex engine is *eager* to return a match. It will not continue backtracking further to see if there is another possible match. It will report the first valid match it finds. Because of greediness, this is the leftmost longest match.

## Laziness Instead of Greediness

The quick fix to this problem is to make the plus *lazy* instead of greedy. Lazy quantifiers are sometimes also called "ungreedy" or "reluctant". You can do that by putting a question markbehind the plus in the regex. You can do the same with the star, the curly braces and the question mark itself. So our example becomes «<.+?>». Let's have another look inside the regex engine.

Again, «<» matches the first „<" in the string. The next token is the dot, this time repeated by a lazy plus. This tells the regex engine to repeat the dot as few times as possible. The minimum is one. So the engine matches the dot with „E". The requirement has been met, and the engine continues with «>» and "M". This fails. Again, the engine will *backtrack*. But this time, the backtracking will force the lazy plus to expand rather than reduce its reach. So the match of «.+» is expanded to „EM", and the engine tries again to continue with «>». Now, „>" is matched successfully. The last token in the regex has been matched. The engine reports that „<EM>" has been successfully matched. That's more like it.

## An Alternative to Laziness

In this case, there is a better option than making the plus lazy. We can use a greedy plus and a negated character class: «<[^>]+>». The reason why this is better is because of the backtracking. When using the lazy plus, the engine has to backtrack for each character in the HTML tag that it is trying to match. When using

the negated character class, no backtracking occurs at all when the string contains valid HTML code. Backtracking slows down the regex engine. You will not notice the difference when doing a single search in a text editor. But you will save plenty of CPU cycles when using such a regex repeatedly in a tight loop in a script that you are writing, or perhaps in a custom syntax coloring scheme for EditPad Pro.

Finally, remember that this tutorial only talks about regex-directed engines. Text-directed engines do not backtrack. They do not get the speed penalty, but they also do not support lazy repetition operators.

## Repeating \Q...\E Escape Sequences

The \Q...\E sequence escapes a string of characters, matching them as literal characters. The escaped characters are treated as individual characters. If you place a quantifier after the `\E`, it will only be applied to the last character. E.g. if you apply «`\Q*\d+*\E+`» to "`*\d+**\d+*`", the match will be „`*\d+**`". Only the asterisk is repeated. Java 4 and 5 have a bug that causes the whole \Q..\E sequence to be repeated, yielding the whole subject string as the match. This was fixed in Java 6.

# 11. Use Round Brackets for Grouping

By placing part of a regular expression inside round brackets or parentheses, you can group that part of the regular expression together. This allows you to apply a regex operator, e.g. a repetition operator, to the entire group. I have already used round brackets for this purpose in previous topics throughout this tutorial.

Note that only round brackets can be used for grouping. Square brackets define a character class, and curly braces are used by a special repetition operator.

## Round Brackets Create a Backreference

Besides grouping part of a regular expression together, round brackets also create a "backreference". A backreference stores the part of the string matched by the part of the regular expression inside the parentheses.

That is, unless you use non-capturing parentheses. Remembering part of the regex match in a backreference, slows down the regex engine because it has more work to do. If you do not use the backreference, you can speed things up by using non-capturing parentheses, at the expense of making your regular expression slightly harder to read.

The regex «`Set(Value)?`» matches „`Set`" or „`SetValue`". In the first case, the first backreference will be empty, because it did not match anything. In the second case, the first backreference will contain „`Value`".

If you do not use the backreference, you can optimize this regular expression into «`Set(?:Value)?`». The question mark and the colon after the opening round bracket are the special syntax that you can use to tell the regex engine that this pair of brackets should not create a backreference. Note the question mark after the opening bracket is unrelated to the question mark at the end of the regex. That question mark is the regex operator that makes the previous token optional. This operator cannot appear after an opening round bracket, because an opening bracket by itself is not a valid regex token. Therefore, there is no confusion between the question mark as an operator to make a token optional, and the question mark as a character to change the properties of a pair of round brackets. The colon indicates that the change we want to make is to turn off capturing the backreference.

## How to Use Backreferences

Backreferences allow you to reuse part of the regex match. You can reuse it inside the regular expression (see below), or afterwards. What you can do with it afterwards, depends on the tool or programming language you are using. The most common usage is in search-and-replace operations. The replacement text will use a special syntax to allow text matched by capturing groups to be reinserted. This syntax differs greatly between various tools and languages, far more than the regex syntax does. Please check the replacement text reference for details.

## Using Backreferences in The Regular Expression

Backreferences can not only be used after a match has been found, but also during the match. Suppose you want to match a pair of opening and closing HTML tags, and the text in between. By putting the opening tag into a backreference, we can reuse the name of the tag for the closing tag. Here's how: «`<([A-Z][A-Z0-9]*)\b[^>]*>.*?</\1>`» . This regex contains only one pair of parentheses, which capture the string matched by «`[A-Z][A-Z0-9]*`» into the first backreference. This backreference is reused with «`\1`» (backslash one). The «/» before it is simply the forward slash in the closing HTML tag that we are trying to match.

To figure out the number of a particular backreference, scan the regular expression from left to right and count the opening round brackets. The first bracket starts backreference number one, the second number two, etc. Non-capturing parentheses are not counted. This fact means that non-capturing parentheses have another benefit: you can insert them into a regular expression without changing the numbers assigned to the backreferences. This can be very useful when modifying a complex regular expression.

You can reuse the same backreference more than once. «`([a-c])x\1x\1`» will match „`axaxa`", „`bxbxb`" and „`cxcxc`".

## Looking Inside The Regex Engine

Let's see how the regex engine applies the above regex to the string "`Testing <B><I>bold italic</I></B> text`". The first token in the regex is the literal «<». The regex engine will traverse the string until it can match at the first „<" in the string. The next token is «`[A-Z]`». The regex engine also takes note that it is now inside the first pair of capturing parentheses. «`[A-Z]`» matches „`B`". The engine advances to «`[A-Z0-9]`» and ">". This match fails. However, because of the star, that's perfectly fine. The position in the string remains at ">". The position in the regex is advanced to «`[^>]`».

This step crosses the closing bracket of the first pair of capturing parentheses. This prompts the regex engine to store what was matched inside them into the first backreference. In this case, „`B`" is stored.

After storing the backreference, the engine proceeds with the match attempt. «`[^>]`» does not match „>". Again, because of another star, this is not a problem. The position in the string remains at ">", and position in the regex is advanced to «>». These obviously match. The next token is a dot, repeated by a lazy star. Because of the laziness, the regex engine will initially skip this token, taking note that it should backtrack in case the remainder of the regex fails.

The engine has now arrived at the second «<» in the regex, and the second "<" in the string. These match. The next token is «/». This does not match "`I`", and the engine is forced to backtrack to the dot. The dot matches the second „<" in the string. The star is still lazy, so the engine again takes note of the available backtracking position and advances to «<» and "`I`". These do not match, so the engine again backtracks.

The backtracking continues until the dot has consumed „`<I>bold italic`". At this point, «<» matches the third „<" in the string, and the next token is «/» which matches "`/`". The next token is «`\1`». Note that the token is the backreference, and not «`B`». The engine does not substitute the backreference in the regular expression. Every time the engine arrives at the backreference, it will read the value that was stored. This means that if the engine had backtracked beyond the first pair of capturing parentheses before arriving the second time at «`\1`», the new value stored in the first backreference would be used. But this did not happen

here, so „B" it is. This fails to match at "I", so the engine backtracks again, and the dot consumes the third "<" in the string.

Backtracking continues again until the dot has consumed „<I>bold italic</I>". At this point, «<» matches „<" and «/» matches „/". The engine arrives again at «\1». The backreference still holds „B". «B» matches „B". The last token in the regex, «>» matches „>". A complete match has been found: „<B><I>bold italic</I></B>".

## Backtracking Into Capturing Groups

You may have wondered about the word boundary «\b» in the «<([A-Z][A-Z0-9]*)\b[^>]*>.*?</\1>» mentioned above. This is to make sure the regex won't match incorrectly paired tags such as "<boo>bold</b>". You may think that cannot happen because the capturing group matches „boo" which causes «\1» to try to match the same, and fail. That is indeed what happens. But then the regex engine backtracks.

Let's take the regex «<([A-Z][A-Z0-9]*)[^>]*>.*?</\1>» without the word boundary and look inside the regex engine at the point where «\1» fails the first time. First, «.*?» continues to expand until it has reached the end of the string, and «</\1>» has failed to match each time «.*?» matched one more character.

Then the regex engine backtracks into the capturing group. «[A-Z0-9]*» has matched „oo", but would just as happily match „o" or nothing at all. When backtracking, «[A-Z0-9]*» is forced to give up one character. The regex engine continues, exiting the capturing group a second time. Since [A-Z][A-Z0-9]* has now matched „bo", that is what is stored into the capturing group, overwriting „boo" that was stored before. «[^>]*» matches the second „o" in the opening tag. «>.*?</» matches „>bold<". «\1» fails again.

The regex engine does all the same backtracking once more, until «[A-Z0-9]*» is forced to give up another character, causing it to match nothing, which the star allows. The capturing group now stores just „b". «[^>]*» now matches „oo". «>.*?</» once again matches „>bold<". «\1» now succeeds, as does «>» and an overall match is found. But not the one we wanted.

There are several solutions to this. One is to use the word boundary. When «[A-Z0-9]*» backtracks the first time, reducing the capturing group to „bo", «\b» fails to match between "o" and "o". This forces «[A-Z0-9]*» to backtrack again immediately. The capturing group is reduced to „b" and the word boundary fails between "b" and "o". There are no further backtracking positions, so the whole match attempt fails.

The reason we need the word boundary is that we're using «[^>]*» to skip over any attributes in the tag. If your paired tags never have any attributes, you can leave that out, and use «<([A-Z][A-Z0-9]*)>.*?</\1>». Each time «[A-Z0-9]*» backtracks, the «>» that follows it will fail to match, quickly ending the match attempt.

If you didn't expect the regex engine to backtrack into capturing groups, you can use an atomic group. The regex engine always backtracks into capturing groups, and never captures atomic groups. You can put the capturing group inside an atomic group to get an atomic capturing group: «(?>(atomic capture))». In this case, we can put the whole opening tag into the atomic group: «(?><([A-Z][A-Z0-9]*)[^>]*>).*?</\1>». The tutorial section on atomic grouping has all the details.

## Backreferences to Failed Groups

The previous section applies to all regex flavors, except those few that don't support capturing groups at all. Flavors behave differently when you start doing things that don't fit the "match the text matched by a previous capturing group" job description.

There is a difference between a backreference to a capturing group that matched nothing, and one to a capturing group that did not participate in the match at all. The regex «`(q?)b\1`» will match „`b`". «`q?`» is optional and matches nothing, causing «`(q?)`» to successfully match and capture nothing. «`b`» matches „`b`" and «`\1`» successfully matches the nothing captured by the group.

The regex «`(q)?b\1`» however will fail to match "`b`". «`(q)`» fails to match at all, so the group never gets to capture anything at all. Because the whole group is optional, the engine does proceed to match «`b`». However, the engine now arrives at «`\1`» which references a group that did not participate in the match attempt at all. This causes the backreference to fail to match at all, mimicking the result of the group. Since there's no «`?`» making «`\1`» optional, the overall match attempt fails.

The only exception is JavaScript. According to the official ECMA standard, a backreference to a non-participating capturing group must successfully match nothing just like a backreference to a participating group that captured nothing does. In other words, in JavaScript, «`(q?)b\1`» and «`(q)?b\1`» both match „`b`".

## Forward References and Invalid References

Modern flavors, notably JGsoft, .NET, Java, Perl, PCRE and Ruby allow forward references. That is: you can use a backreference to a group that appears later in the regex. Forward references are obviously only useful if they're inside a repeated group. Then there can be situations in which the regex engine evaluates the backreference after the group has already matched. Before the group is attempted, the backreference will fail like a backreference to a failed group does.

If forward references are supported, the regex «`(\2two|(one))+`» will match „`oneonetwo`". At the start of the string, «`\2`» fails. Trying the other alternative, „`one`" is matched by the second capturing group, and subsequently by the first group. The first group is then repeated. This time, «`\2`» matches „`one`" as captured by the second group. «`two`» then matches „`two`". With two repetitions of the first group, the regex has matched the whole subject string.

A nested reference is a backreference inside the capturing group that it references, e.g. «`(\1two|(one))+`». This regex will give exactly the same behavior with flavors that support forward references. Some flavors that don't support forward references do support nested references. This includes JavaScript.

With all other flavors, using a backreference before its group in the regular expression is the same as using a backreference to a group that doesn't exist at all. All flavors discussed in this tutorial, except JavaScript and Ruby, treat backreferences to undefined groups as an error. In JavaScript and Ruby, they always result in a zero-width match. For Ruby this is a potential pitfall. In Ruby, «`(a)(b)?\2`» will fail to match "`a`", because «`\2`» references a non-participating group. But «`(a)(b)?\7`» will match „`a`". For JavaScript this is logical, as backreferences to non-participating groups do the same. Both regexes will match „`a`".

## Repetition and Backreferences

As I mentioned in the above inside look, the regex engine does not permanently substitute backreferences in the regular expression. It will use the last match saved into the backreference each time it needs to be used. If a new match is found by capturing parentheses, the previously saved match is overwritten. There is a clear difference between «`([abc]+)`» and «`([abc])+`». Though both successfully match „`cab`", the first regex will put „`cab`" into the first backreference, while the second regex will only store „`b`". That is because in the second regex, the plus caused the pair of parentheses to repeat three times. The first time, „`c`" was stored. The second time „`a`" and the third time „`b`". Each time, the previous value was overwritten, so „`b`" remains.

This also means that «`([abc]+)=\1`» will match „`cab=cab`", and that «`([abc])+=\1`» will not. The reason is that when the engine arrives at «`\1`», it holds «`b`» which fails to match "`c`". Obvious when you look at a simple example like this one, but a common cause of difficulty with regular expressions nonetheless. When using backreferences, always double check that you are really capturing what you want.

## Useful Example: Checking for Doubled Words

When editing text, doubled words such as "the the" easily creep in. Using the regex «`\b(\w+)\s+\1\b`» in your text editor, you can easily find them. To delete the second word, simply type in "`\1`" as the replacement text and click the Replace button.

## Parentheses and Backreferences Cannot Be Used Inside Character Classes

Round brackets cannot be used inside character classes, at least not as metacharacters. When you put a round bracket in a character class, it is treated as a literal character. So the regex «`[(a)b]`» matches „`a`", „`b`", „`(`" and „`)`".

Backreferences also cannot be used inside a character class. The \1 in regex like «`(a)[\1b]`» will be interpreted as an octal escape in most regex flavors. So this regex will match an „`a`" followed by either «`\x01`» or a «`b`».

# 12. Named Capturing Groups

All modern regular expression engines support capturing groups, which are numbered from left to right, starting with one. The numbers can then be used in backreferences to match the same text again in the regular expression, or to use part of the regex match for further processing. In a complex regular expression with many capturing groups, the numbering can get a little confusing.

## Named Capture with Python, PCRE and PHP

Python's regex module was the first to offer a solution: named capture. By assigning a name to a capturing group, you can easily reference it by name. «`(?P<name>group)`» captures the match of «`group`» into the backreference "name". You can reference the contents of the group with the numbered backreference «`\1`» or the named backreference «`(?P=name)`».

The open source PCRE library has followed Python's example, and offers named capture using the same syntax. The PHP preg functions offer the same functionality, since they are based on PCRE.

Python's `sub()` function allows you to reference a named group as "`\1`" or "`\g<name>`". This does *not* work in PHP. In PHP, you can use double-quoted string interpolation with the `$regs` parameter you passed to `pcre_match()`: "`$regs['name']`".

## Named Capture with .NET's System.Text.RegularExpressions

The regular expression classes of the .NET framework also support named capture. Unfortunately, the Microsoft developers decided to invent their own syntax, rather than follow the one pioneered by Python. Currently, no other regex flavor supports Microsoft's version of named capture.

Here is an example with two capturing groups in .NET style: «`(?<first>group)(?'second'group)`». As you can see, .NET offers two syntaxes to create a capturing group: one using sharp brackets, and the other using single quotes. The first syntax is preferable in strings, where single quotes may need to be escaped. The second syntax is preferable in ASP code, where the sharp brackets are used for HTML tags. You can use the pointy bracket flavor and the quoted flavors interchangeably.

To reference a capturing group inside the regex, use «`\k<name>`» or «`\k'name'`». Again, you can use the two syntactic variations interchangeably.

When doing a search-and-replace, you can reference the named group with the familiar dollar sign syntax: "`${name}`". Simply use a name instead of a number between the curly braces.

## Multiple Groups with The Same Name

The .NET framework allows multiple groups in the regular expression to have the same name. If you do so, both groups will store their matches in the same Group object. You won't be able to distinguish which group captured the text. This can be useful in regular expressions with multiple alternatives to match the same thing. E.g. if you want to match "a" followed by a digit 0..5, or "b" followed by a digit 4..7, and you only care about

the digit, you could use the regex «`a(?'digit'[0-5])|b(?'digit'[4-7])`». The group named "digit" will then give you the digit 0..7 that was matched, regardless of the letter.

Python and PCRE do not allow multiple groups to use the same name. Doing so will give a regex compilation error.

## Names and Numbers for Capturing Groups

Here is where things get a bit ugly. Python and PCRE treat named capturing groups just like unnamed capturing groups, and number both kinds from left to right, starting with one. The regex «`(a)(?P<x>b)(c)(?P<y>d)`» matches „`abcd`" as expected. If you do a search-and-replace with this regex and the replacement "`\1\2\3\4`", you will get "`abcd`". All four groups were numbered from left to right, from one till four. Easy and logical.

Things are quite a bit more complicated with the .NET framework. The regex «`(a)(?<x>b)(c)(?<y>d)`» again matches „`abcd`". However, if you do a search-and-replace with "`$1$2$3$4`" as the replacement, you will get "`acbd`". Probably not what you expected.

The .NET framework *does* number named capturing groups from left to right, but numbers them *after* all the unnamed groups have been numbered. So the unnamed groups «`(a)`» and «`(c)`» get numbered first, from left to right, starting at one. Then the named groups «`(?<x>b)`» and «`(?<y>d)`» get their numbers, continuing from the unnamed groups, in this case: three.

To make things simple, when using .NET's regex support, just assume that named groups do not get numbered at all, and reference them by name exclusively. To keep things compatible across regex flavors, I strongly recommend that you do not mix named and unnamed capturing groups at all. Either give a group a name, or make it non-capturing as in «`(?:nocapture)`». Non-capturing groups are more efficient, since the regex engine does not need to keep track of their matches.

## Best of Both Worlds

The JGsoft regex engine supports both .NET-style and Python-style named capture. Python-style named groups are numbered along unnamed ones, like Python does. .NET-style named groups are numbered afterwards, like .NET does. You can mix both styles in the same regex. The JGsoft engine allows multiple groups to use the same name, regardless of the syntax used.

In PowerGREP, named capturing groups play a special role. Groups with the same name are shared between all regular expressions and replacement texts in the same PowerGREP action. This allows captured by a named capturing group in one part of the action to be referenced in a later part of the action. Because of this, PowerGREP does not allow numbered references to named capturing groups at all. When mixing named and numbered groups in a regex, the numbered groups are still numbered following the Python and .NET rules, like the JGsoft flavor always does.

# 13. Unicode Regular Expressions

Unicode is a character set that aims to define all characters and glyphs from all human languages, living and dead. With more and more software being required to support multiple languages, or even just *any* language, Unicode has been strongly gaining popularity in recent years. Using different character sets for different languages is simply too cumbersome for programmers and users.

Unfortunately, Unicode brings its own requirements and pitfalls when it comes to regular expressions. Of the regex flavors discussed in this tutorial, Java, XML and the .NET framework use Unicode-based regex engines. Perl supports Unicode starting with version 5.6. PCRE can optionally be compiled with Unicode support. Note that PCRE is far less flexible in what it allows for the `\p` tokens, despite its name "Perl-compatible". The PHP preg functions, which are based on PCRE, support Unicode when the /u option is appended to the regular expression.

RegexBuddy's regex engine is fully Unicode-based starting with version 2.0.0. RegexBuddy 1.x.x did not support Unicode at all. PowerGREP uses the same Unicode regex engine starting with version 3.0.0. Earlier versions would convert Unicode files to ANSI prior to grepping with an 8-bit (i.e. non-Unicode) regex engine. EditPad Pro supports Unicode starting with version 6.0.0.

## Characters, Code Points and Graphemes or How Unicode Makes a Mess of Things

Most people would consider "à" a single character. Unfortunately, it need not be depending on the meaning of the word "character".

All Unicode regex engines discussed in this tutorial treat any single Unicode *code point* as a single character. When this tutorial tells you that the dot matches any single character, this translates into Unicode parlance as "the dot matches any single Unicode code point". In Unicode, "à" can be encoded as two code points: U+0061 (a) followed by U+0300 (grave accent). In this situation, «.» applied to "à" will match „a" without the accent. «^.$» will fail to match, since the string consists of two code points. «^..$» matches „à".

The Unicode code point U+0300 (grave accent) is a *combining mark*. Any code point that is not a combining mark can be followed by any number of combining marks. This sequence, like U+0061 U+0300 above, is displayed as a single *grapheme* on the screen.

Unfortunately, "à" can also be encoded with the single Unicode code point U+00E0 (a with grave accent). The reason for this duality is that many historical character sets encode "a with grave accent" as a single character. Unicode's designers thought it would be useful to have a one-on-one mapping with popular legacy character sets, in addition to the Unicode way of separating marks and base letters (which makes arbitrary combinations not supported by legacy character sets possible).

## How to Match a Single Unicode Grapheme

Matching a single grapheme, whether it's encoded as a single code point, or as multiple code points using combining marks, is easy in Perl, RegexBuddy and PowerGREP: simply use «\X». You can consider «\X» the Unicode version of the dot in regex engines that use plain ASCII. There is one difference, though: «\X»

always matches line break characters, whereas the dot does not match line break characters unless you enable the dot matches newline matching mode.

Java and .NET unfortunately do not support «\X» (yet). Use «\P{M}\p{M}*+» or «(?>\P{M}\p{M}*)» as a reasonably close substitute. To match any number of graphemes, use «(?>\P{M}\p{M}*)+» as a substitute for «\X+».

## Matching a Specific Code Point

To match a specific Unicode code point, use «\uFFFF» where FFFF is the hexadecimal number of the code point you want to match. You must always specify 4 hexadecimal digits E.g. «\u00E0» matches „à", but only when encoded as a single code point U+00E0.

Perl and PCRE do not support the «\uFFFF» syntax. They use «\x{FFFF}» instead. You can omit leading zeros in the hexadecimal number between the curly braces. Since \x by itself is not a valid regex token, «\x{1234}» can never be confused to match \x 1234 times. It always matches the Unicode code point U+1234. «\x{1234}{5678}» will try to match code point U+1234 exactly 5678 times.

In Java, the regex token «\uFFFF» only matches the specified code point, even when you turned on canonical equivalence. However, the same syntax \uFFFF is also used to insert Unicode characters into literal strings in the Java source code. `Pattern.compile("\u00E0")` will match both the single-code-point and double-code-point encodings of „à", while `Pattern.compile("\\u00E0")` matches only the single-code-point version. Remember that when writing a regex as a Java string literal, backslashes must be escaped. The former Java code compiles the regex «à», while the latter compiles «\u00E0». Depending on what you're doing, the difference may be significant.

JavaScript, which does not offer any Unicode support through its RegExp class, does support «\uFFFF» for matching a single Unicode code point as part of its string syntax.

XML Schema does not have a regex token for matching Unicode code points. However, you can easily use XML entities like &#xFFFF; to insert literal code points into your regular expression.

## Unicode Character Properties

In addition to complications, Unicode also brings new possibilities. One is that each Unicode character belongs to a certain category. You can match a single character belonging to a particular category with «\p{}». You can match a single character *not* belonging to a particular category with «\P{}».

Again, "character" really means "Unicode code point". «\p{L}» matches a single code point in the category "letter". If your input string is "à" encoded as U+0061 U+0300, it matches „a" without the accent. If the input is "à" encoded as U+00E0, it matches „à" with the accent. The reason is that both the code points U+0061 (a) and U+00E0 (à) are in the category "letter", while U+0300 is in the category "mark".

You should now understand why «\P{M}\p{M}*» is the equivalent of «\X». «\P{M}» matches a code point that is not a combining mark, while «\p{M}*» matches zero or more code points that are combining marks. To match a letter including any diacritics, use «\p{L}\p{M}*». This last regex will always match „à", regardless of how it is encoded.

The .NET Regex class and PCRE are case sensitive when it checks the part between curly braces of a `\p` token. «`\p{Zs}`» will match any kind of space character, while «`\p{zs}`» will throw an error. All other regex engines described in this tutorial will match the space in both cases, ignoring the case of the property between the curly braces. Still, I recommend you make a habit of using the same uppercase and lowercase combination as I did in the list of properties below. This will make your regular expressions work with all Unicode regex engines.

In addition to the standard notation, «`\p{L}`», Java, Perl, PCRE and the JGsoft engine allow you to use the shorthand «`\pL`». The shorthand only works with single-letter Unicode properties. «`\pLl`» is *not* the equivalent of «`\p{Ll}`». It is the equivalent of «`\p{L}l`» which matches „`Al`" or „`àl`" or any Unicode letter followed by a literal „`l`".

Perl and the JGsoft engine also support the longhand «`\p{Letter}`». You can find a complete list of all Unicode properties below. You may omit the underscores or use hyphens or spaces instead.

- «`\p{L}`» or «`\p{Letter}`»: any kind of letter from any language.
  - «`\p{Ll}`» or «`\p{Lowercase_Letter}`»: a lowercase letter that has an uppercase variant.
  - «`\p{Lu}`» or «`\p{Uppercase_Letter}`»: an uppercase letter that has a lowercase variant.
  - «`\p{Lt}`» or «`\p{Titlecase_Letter}`»: a letter that appears at the start of a word when only the first letter of the word is capitalized.
  - «`\p{L&}`» or «`\p{Letter&}`»: a letter that exists in lowercase and uppercase variants (combination of Ll, Lu and Lt).
  - «`\p{Lm}`» or «`\p{Modifier_Letter}`»: a special character that is used like a letter.
  - «`\p{Lo}`» or «`\p{Other_Letter}`»: a letter or ideograph that does not have lowercase and uppercase variants.
- «`\p{M}`» or «`\p{Mark}`»: a character intended to be combined with another character (e.g. accents, umlauts, enclosing boxes, etc.).
  - «`\p{Mn}`» or «`\p{Non_Spacing_Mark}`»: a character intended to be combined with another character without taking up extra space (e.g. accents, umlauts, etc.).
  - «`\p{Mc}`» or «`\p{Spacing_Combining_Mark}`»: a character intended to be combined with another character that takes up extra space (vowel signs in many Eastern languages).
  - «`\p{Me}`» or «`\p{Enclosing_Mark}`»: a character that encloses the character is is combined with (circle, square, keycap, etc.).
- «`\p{Z}`» or «`\p{Separator}`»: any kind of whitespace or invisible separator.
  - «`\p{Zs}`» or «`\p{Space_Separator}`»: a whitespace character that is invisible, but does take up space.
  - «`\p{Zl}`» or «`\p{Line_Separator}`»: line separator character U+2028.
  - «`\p{Zp}`» or «`\p{Paragraph_Separator}`»: paragraph separator character U+2029.
- «`\p{S}`» or «`\p{Symbol}`»: math symbols, currency signs, dingbats, box-drawing characters, etc..
  - «`\p{Sm}`» or «`\p{Math_Symbol}`»: any mathematical symbol.
  - «`\p{Sc}`» or «`\p{Currency_Symbol}`»: any currency sign.
  - «`\p{Sk}`» or «`\p{Modifier_Symbol}`»: a combining character (mark) as a full character on its own.
  - «`\p{So}`» or «`\p{Other_Symbol}`»: various symbols that are not math symbols, currency signs, or combining characters.
- «`\p{N}`» or «`\p{Number}`»: any kind of numeric character in any script.
  - «`\p{Nd}`» or «`\p{Decimal_Digit_Number}`»: a digit zero through nine in any script except ideographic scripts.
  - «`\p{Nl}`» or «`\p{Letter_Number}`»: a number that looks like a letter, such as a Roman numeral.

    - «\p{No}» or «\p{Other_Number}»: a superscript or subscript digit, or a number that is not a digit 0..9 (excluding numbers from ideographic scripts).
- «\p{P}» or «\p{Punctuation}»: any kind of punctuation character.
    - «\p{Pd}» or «\p{Dash_Punctuation}»: any kind of hyphen or dash.
    - «\p{Ps}» or «\p{Open_Punctuation}»: any kind of opening bracket.
    - «\p{Pe}» or «\p{Close_Punctuation}»: any kind of closing bracket.
    - «\p{Pi}» or «\p{Initial_Punctuation}»: any kind of opening quote.
    - «\p{Pf}» or «\p{Final_Punctuation}»: any kind of closing quote.
    - «\p{Pc}» or «\p{Connector_Punctuation}»: a punctuation character such as an underscore that connects words.
    - «\p{Po}» or «\p{Other_Punctuation}»: any kind of punctuation character that is not a dash, bracket, quote or connector.
- «\p{C}» or «\p{Other}»: invisible control characters and unused code points.
    - «\p{Cc}» or «\p{Control}»: an ASCII 0x00..0x1F or Latin-1 0x80..0x9F control character.
    - «\p{Cf}» or «\p{Format}»: invisible formatting indicator.
    - «\p{Co}» or «\p{Private_Use}»: any code point reserved for private use.
    - «\p{Cs}» or «\p{Surrogate}»: one half of a surrogate pair in UTF-16 encoding.
    - «\p{Cn}» or «\p{Unassigned}»: any code point to which no character has been assigned.

## Unicode Scripts

The Unicode standard places each assigned code point (character) into one script. A script is a group of code points used by a particular human writing system. Some scripts like `Thai` correspond with a single human language. Other scripts like `Latin` span multiple languages.

Some languages are composed of multiple scripts. There is no Japanese Unicode script. Instead, Unicode offers the `Hiragana`, `Katakana`, `Han` and `Latin` scripts that Japanese documents are usually composed of.

A special script is the `Common` script. This script contains all sorts of characters that are common to a wide range of scripts. It includes all sorts of punctuation, whitespace and miscellaneous symbols.

All assigned Unicode code points (those matched by «\P{Cn}») are part of exactly one Unicode script. All unassigned Unicode code points (those matched by «\p{Cn}») are not part of any Unicode script at all.

Very few regular expression engines support Unicode scripts today. Of all the flavors discussed in this tutorial, only the JGsoft engine, Perl and PCRE can match Unicode scripts. Here's a complete list of all Unicode scripts:

1. «\p{Common}»
2. «\p{Arabic}»
3. «\p{Armenian}»
4. «\p{Bengali}»
5. «\p{Bopomofo}»
6. «\p{Braille}»
7. «\p{Buhid}»
8. «\p{CanadianAboriginal}»
9. «\p{Cherokee}»
10. «\p{Cyrillic}»
11. «\p{Devanagari}»

12. «\p{Ethiopic}»
13. «\p{Georgian}»
14. «\p{Greek}»
15. «\p{Gujarati}»
16. «\p{Gurmukhi}»
17. «\p{Han}»
18. «\p{Hangul}»
19. «\p{Hanunoo}»
20. «\p{Hebrew}»
21. «\p{Hiragana}»
22. «\p{Inherited}»
23. «\p{Kannada}»
24. «\p{Katakana}»
25. «\p{Khmer}»
26. «\p{Lao}»
27. «\p{Latin}»
28. «\p{Limbu}»
29. «\p{Malayalam}»
30. «\p{Mongolian}»
31. «\p{Myanmar}»
32. «\p{Ogham}»
33. «\p{Oriya}»
34. «\p{Runic}»
35. «\p{Sinhala}»
36. «\p{Syriac}»
37. «\p{Tagalog}»
38. «\p{Tagbanwa}»
39. «\p{TaiLe}»
40. «\p{Tamil}»
41. «\p{Telugu}»
42. «\p{Thaana}»
43. «\p{Thai}»
44. «\p{Tibetan}»
45. «\p{Yi}»

Instead of the «\p{Latin}» syntax you can also use «\p{IsLatin}». The "Is" syntax is useful for distinguishing between scripts and blocks, as explained in the next section. Unfortunately, PCRE does not support "Is" as of this writing.

## Unicode Blocks

The Unicode standard divides the Unicode character map into different blocks or ranges of code points. Each block is used to define characters of a particular script like "Tibetan" or belonging to a particular group like "Braille Patterns". Most blocks include unassigned code points, reserved for future expansion of the Unicode standard.

Note that Unicode blocks do not correspond 100% with scripts. An essential difference between blocks and scripts is that a block is a single contiguous range of code points, as listed below. Scripts consist of characters taken from all over the Unicode character map. Blocks may include unassigned code points (i.e. code points

matched by «\p{Cn}»). Scripts never include unassigned code points. Generally, if you're not sure whether to use a Unicode script or Unicode block, use the script.

E.g. the Currency block does not include the dollar and yen symbols. Those are found in the Basic_Latin and Latin-1_Supplement blocks instead, for historical reasons, even though both are currency symbols, and the yen symbol is not a Latin character. You should not blindly use any of the blocks listed below based on their names. Instead, look at the ranges of characters they actually match. A tool like RegexBuddy can be very helpful with this. E.g. the Unicode property «\p{Sc}» or «\p{Currency_Symbol}» would be a better choice than the Unicode block «\p{InCurrency}» when trying to find all currency symbols.

1. «\p{InBasic_Latin}»: U+0000..U+007F
2. «\p{InLatin-1_Supplement}»: U+0080..U+00FF
3. «\p{InLatin_Extended-A}»: U+0100..U+017F
4. «\p{InLatin_Extended-B}»: U+0180..U+024F
5. «\p{InIPA_Extensions}»: U+0250..U+02AF
6. «\p{InSpacing_Modifier_Letters}»: U+02B0..U+02FF
7. «\p{InCombining_Diacritical_Marks}»: U+0300..U+036F
8. «\p{InGreek_and_Coptic}»: U+0370..U+03FF
9. «\p{InCyrillic}»: U+0400..U+04FF
10. «\p{InCyrillic_Supplementary}»: U+0500..U+052F
11. «\p{InArmenian}»: U+0530..U+058F
12. «\p{InHebrew}»: U+0590..U+05FF
13. «\p{InArabic}»: U+0600..U+06FF
14. «\p{InSyriac}»: U+0700..U+074F
15. «\p{InThaana}»: U+0780..U+07BF
16. «\p{InDevanagari}»: U+0900..U+097F
17. «\p{InBengali}»: U+0980..U+09FF
18. «\p{InGurmukhi}»: U+0A00..U+0A7F
19. «\p{InGujarati}»: U+0A80..U+0AFF
20. «\p{InOriya}»: U+0B00..U+0B7F
21. «\p{InTamil}»: U+0B80..U+0BFF
22. «\p{InTelugu}»: U+0C00..U+0C7F
23. «\p{InKannada}»: U+0C80..U+0CFF
24. «\p{InMalayalam}»: U+0D00..U+0D7F
25. «\p{InSinhala}»: U+0D80..U+0DFF
26. «\p{InThai}»: U+0E00..U+0E7F
27. «\p{InLao}»: U+0E80..U+0EFF
28. «\p{InTibetan}»: U+0F00..U+0FFF
29. «\p{InMyanmar}»: U+1000..U+109F
30. «\p{InGeorgian}»: U+10A0..U+10FF
31. «\p{InHangul_Jamo}»: U+1100..U+11FF
32. «\p{InEthiopic}»: U+1200..U+137F
33. «\p{InCherokee}»: U+13A0..U+13FF
34. «\p{InUnified_Canadian_Aboriginal_Syllabics}»: U+1400..U+167F
35. «\p{InOgham}»: U+1680..U+169F
36. «\p{InRunic}»: U+16A0..U+16FF
37. «\p{InTagalog}»: U+1700..U+171F
38. «\p{InHanunoo}»: U+1720..U+173F
39. «\p{InBuhid}»: U+1740..U+175F
40. «\p{InTagbanwa}»: U+1760..U+177F
41. «\p{InKhmer}»: U+1780..U+17FF

42. «\p{InMongolian}»: U+1800..U+18AF
43. «\p{InLimbu}»: U+1900..U+194F
44. «\p{InTai_Le}»: U+1950..U+197F
45. «\p{InKhmer_Symbols}»: U+19E0..U+19FF
46. «\p{InPhonetic_Extensions}»: U+1D00..U+1D7F
47. «\p{InLatin_Extended_Additional}»: U+1E00..U+1EFF
48. «\p{InGreek_Extended}»: U+1F00..U+1FFF
49. «\p{InGeneral_Punctuation}»: U+2000..U+206F
50. «\p{InSuperscripts_and_Subscripts}»: U+2070..U+209F
51. «\p{InCurrency_Symbols}»: U+20A0..U+20CF
52. «\p{InCombining_Diacritical_Marks_for_Symbols}»: U+20D0..U+20FF
53. «\p{InLetterlike_Symbols}»: U+2100..U+214F
54. «\p{InNumber_Forms}»: U+2150..U+218F
55. «\p{InArrows}»: U+2190..U+21FF
56. «\p{InMathematical_Operators}»: U+2200..U+22FF
57. «\p{InMiscellaneous_Technical}»: U+2300..U+23FF
58. «\p{InControl_Pictures}»: U+2400..U+243F
59. «\p{InOptical_Character_Recognition}»: U+2440..U+245F
60. «\p{InEnclosed_Alphanumerics}»: U+2460..U+24FF
61. «\p{InBox_Drawing}»: U+2500..U+257F
62. «\p{InBlock_Elements}»: U+2580..U+259F
63. «\p{InGeometric_Shapes}»: U+25A0..U+25FF
64. «\p{InMiscellaneous_Symbols}»: U+2600..U+26FF
65. «\p{InDingbats}»: U+2700..U+27BF
66. «\p{InMiscellaneous_Mathematical_Symbols-A}»: U+27C0..U+27EF
67. «\p{InSupplemental_Arrows-A}»: U+27F0..U+27FF
68. «\p{InBraille_Patterns}»: U+2800..U+28FF
69. «\p{InSupplemental_Arrows-B}»: U+2900..U+297F
70. «\p{InMiscellaneous_Mathematical_Symbols-B}»: U+2980..U+29FF
71. «\p{InSupplemental_Mathematical_Operators}»: U+2A00..U+2AFF
72. «\p{InMiscellaneous_Symbols_and_Arrows}»: U+2B00..U+2BFF
73. «\p{InCJK_Radicals_Supplement}»: U+2E80..U+2EFF
74. «\p{InKangxi_Radicals}»: U+2F00..U+2FDF
75. «\p{InIdeographic_Description_Characters}»: U+2FF0..U+2FFF
76. «\p{InCJK_Symbols_and_Punctuation}»: U+3000..U+303F
77. «\p{InHiragana}»: U+3040..U+309F
78. «\p{InKatakana}»: U+30A0..U+30FF
79. «\p{InBopomofo}»: U+3100..U+312F
80. «\p{InHangul_Compatibility_Jamo}»: U+3130..U+318F
81. «\p{InKanbun}»: U+3190..U+319F
82. «\p{InBopomofo_Extended}»: U+31A0..U+31BF
83. «\p{InKatakana_Phonetic_Extensions}»: U+31F0..U+31FF
84. «\p{InEnclosed_CJK_Letters_and_Months}»: U+3200..U+32FF
85. «\p{InCJK_Compatibility}»: U+3300..U+33FF
86. «\p{InCJK_Unified_Ideographs_Extension_A}»: U+3400..U+4DBF
87. «\p{InYijing_Hexagram_Symbols}»: U+4DC0..U+4DFF
88. «\p{InCJK_Unified_Ideographs}»: U+4E00..U+9FFF
89. «\p{InYi_Syllables}»: U+A000..U+A48F
90. «\p{InYi_Radicals}»: U+A490..U+A4CF
91. «\p{InHangul_Syllables}»: U+AC00..U+D7AF
92. «\p{InHigh_Surrogates}»: U+D800..U+DB7F
93. «\p{InHigh_Private_Use_Surrogates}»: U+DB80..U+DBFF

94. «\p{InLow_Surrogates}»: U+DC00..U+DFFF
95. «\p{InPrivate_Use_Area}»: U+E000..U+F8FF
96. «\p{InCJK_Compatibility_Ideographs}»: U+F900..U+FAFF
97. «\p{InAlphabetic_Presentation_Forms}»: U+FB00..U+FB4F
98. «\p{InArabic_Presentation_Forms-A}»: U+FB50..U+FDFF
99. «\p{InVariation_Selectors}»: U+FE00..U+FE0F
100. «\p{InCombining_Half_Marks}»: U+FE20..U+FE2F
101. «\p{InCJK_Compatibility_Forms}»: U+FE30..U+FE4F
102. «\p{InSmall_Form_Variants}»: U+FE50..U+FE6F
103. «\p{InArabic_Presentation_Forms-B}»: U+FE70..U+FEFF
104. «\p{InHalfwidth_and_Fullwidth_Forms}»: U+FF00..U+FFEF
105. «\p{InSpecials}»: U+FFF0..U+FFFF

Not all Unicode regex engines use the same syntax to match Unicode blocks. Perl and Java use the «\p{InBlock}» syntax as listed above. .NET and XML use «\p{IsBlock}» instead. The JGsoft engine supports both notations. I recommend you use the "In" notation if your regex engine supports it. "In" can only be used for Unicode blocks, while "Is" can also be used for Unicode properties and scripts, depending on the regular expression flavor you're using. By using "In", it's obvious you're matching a block and not a similarly named property or script.

In .NET and XML, you must omit the underscores but keep the hyphens in the block names. E.g. Use «\p{IsLatinExtended-A}» instead of «\p{InLatin_Extended-A}». Perl and Java allow you to use an underscore, hyphen, space or nothing for each underscore or hyphen in the block's name. .NET and XML also compare the names case sensitively, while Perl and Java do not. «\p{islatinextended-a}» throws an error in .NET, while «\p{inlatinextended-a}» works fine in Perl and Java.

The JGsoft engine supports all of the above notations. You can use "In" or "Is", ignore differences in upper and lower case, and use spaces, underscores and hyphens as you like. This way you can keep using the syntax of your favorite programming language, and have it work as you'd expect in PowerGREP or EditPad Pro.

The actual names of the blocks are the same in all regular expression engines. The block names are defined in the Unicode standard. PCRE does not support Unicode blocks.

## Alternative Unicode Regex Syntax

Unicode is a relatively new addition to the world of regular expressions. As you guessed from my explanations of different notations, different regex engine designers unfortunately have different ideas about the syntax to use. Perl and Java even support a few additional alternative notations that you may encounter in regular expressions created by others. I recommend against using these notations in your own regular expressions, to maintain clarity and compatibility with other regex flavors, and understandability by people more familiar with other flavors.

If you are just getting started with Unicode regular expressions, you may want to skip this section until later, to avoid confusion (if the above didn't confuse you already).

In Perl and PCRE regular expressions, you may encounter a Unicode property like «\p{^Lu}» or «\p{^Letter}». These are negated properties identical to «\P{Lu}» or «\P{Letter}». Since very few regex flavors support the «\p{^L}» notation, and all Unicode-compatible regex flavors (including Perl and PCRE) support «\P{L}», I strongly recommend you use the latter syntax.

Perl (but not PCRE) and Java support the «`\p{IsL}`» notation, prefixing one-letter and two-letter Unicode property notations with "Is". Since very few regex flavors support the «`\p{IsL}`» notation, and all Unicode-compatible regex flavors (including Perl and Java) support «`\p{L}`», I strongly recommend you use the latter syntax.

Perl and Java allow you to omit the "In" when matching Unicode blocks, so you can write «`\p{Arrows}`» instead of «`\p{InArrows}`». Perl can also match Unicode scripts, and some scripts like "Hebrew" have the same name as a Unicode block. In that situation, Perl will match the Hebrew script instead of the Hebrew block when you write «`\p{Hebrew}`». While there are no Unicode properties with the same names as blocks, the property «`\p{Currency_Symbol}`» is confusingly similar to the block «`\p{Currency}`». As I explained in the section on Unicode blocks, the characters they match are quite different. To avoid all such confusion, I strongly recommend you use the "In" syntax for blocks, the "Is" syntax for scripts (if supported), and the shorthand syntax «`\p{Lu}`» for properties.

Again, the JGsoft engine supports all of the above oddball notations. This is only done to allow you to copy and paste regular expressions and have them work as they do in Perl or Java. You should consider these notations deprecated.

## Do You Need To Worry About Different Encodings?

While you should always keep in mind the pitfalls created by the different ways in which accented characters can be encoded, you don't always have to worry about them. If you know that your input string and your regex use the same style, then you don't have to worry about it at all. This process is called Unicode *normalization*. All programming languages with native Unicode support, such as Java, C# and VB.NET, have library routines for normalizing strings. If you normalize both the subject and regex before attempting the match, there won't be any inconsistencies.

If you are using Java, you can pass the CANON_EQ flag as the second parameter to Pattern.compile(). This tells the Java regex engine to consider *canonically equivalent* characters as identical. E.g. the regex «à» encoded as U+00E0 will match „à" encoded as U+0061 U+0300, and vice versa. None of the other regex engines currently support canonical equivalence while matching.

If you type the à key on the keyboard, all word processors that I know of will insert the code point U+00E0 into the file. So if you're working with text that you typed in yourself, any regex that you type in yourself will match in the same way.

Finally, if you're using PowerGREP to search through text files encoded using a traditional Windows (often called "ANSI") or ISO-8859 code page, PowerGREP will always use the one-on-one substitution. Since all the Windows or ISO-8859 code pages encode accented characters as a single code point, all software that I know of will use a single Unicode code point for each character when converting the file to Unicode.

# 14. Regex Matching Modes

Most regular expression engines discussed in this tutorial support the following four matching modes:

- `/i` makes the regex match case insensitive.
- `/s` enables "single-line mode". In this mode, the dot matches newlines.
- `/m` enables "multi-line mode". In this mode, the caret and dollar match before and after newlines in the subject string.
- `/x` enables “free-spacing mode”. In this mode, whitespace between regex tokens is ignored, and an unescaped # starts a comment.

Two languages that don’t support all of the above four are JavaScript and Ruby. Some regex flavors also have additional modes or options that have single letter equivalents. These are very implementation-dependent.

Most tools that support regular expressions have checkboxes or similar controls that you can use to turn these modes on or off. Most programming languages allow you to pass option flags when constructing the regex object. E.g. in Perl, `m/regex/i` turns on case insensitivity, while `Pattern.compile("regex", Pattern.CASE_INSENSITIVE)` does the same in Java.

## Specifying Modes Inside The Regular Expression

Sometimes, the tool or language does not provide the ability to specify matching options. E.g. the handy `String.matches()` method in Java does not take a parameter for matching options like `Pattern.compile()` does.

In that situation, you can add a mode modifier to the start of the regex. E.g. `(?i)` turns on case insensitivity, while `(?ism)` turns on all three options.

## Turning Modes On and Off for Only Part of The Regular Expression

Modern regex flavors allow you to apply modifiers to only part of the regular expression. If you insert the modifier `(?ism)` in the middle of the regex, the modifier only applies to the part of the regex to the right of the modifier. You can turn off modes by preceding them with a minus sign. All modes after the minus sign will be turned off. E.g. `(?i-sm)` turns on case insensitivity, and turns off both single-line mode and multi-line mode.

Not all regex flavors support this. JavaScript and Python apply all mode modifiers to the entire regular expression. They don’t support the `(?-ismx)` syntax, since turning off an option is pointless when mode modifiers apply to the whole regular expressions. All options are off by default.

You can quickly test how the regex flavor you’re using handles mode modifiers. The regex «`(?i)te(?-i)st`» should match „`test`” and „`TEst`”, but not “`teST`” or “`TEST`”.

## Modifier Spans

Instead of using two modifiers, one to turn an option on, and one to turn it off, you use a modifier span. «`(?i)ignorecase(?-i)casesensitive(?i)ignorecase`» is equivalent to «`(?i)ignorecase(?-i:casesensitive)ignorecase`». You have probably noticed the resemblance between the modifier span and the non-capturing group «`(?:group)`». Technically, the non-capturing group is a modifier span that does not change any modifiers. It is obvious that the modifier span does not create a backreference.

Modifier spans are supported by all regex flavors that allow you to use mode modifiers in the middle of the regular expression, and by those flavors only. These include the JGsoft engine, .NET, Java, Perl and PCRE.

# 15. Possessive Quantifiers

When discussing the repetition operators or quantifiers, I explained the difference between greedy and lazy repetition. Greediness and laziness determine the order in which the regex engine tries the possible permutations of the regex pattern. A greedy quantifier will first try to repeat the token as many times as possible, and gradually give up matches as the engine backtracks to find an overall match. A lazy quantifier will first repeat the token as few times as required, and gradually expand the match as the engine backtracks through the regex to find an overall match.

Because greediness and laziness change the order in which permutations are tried, they can change the overall regex match. However, they do not change the fact that the regex engine will backtrack to try all possible permutations of the regular expression in case no match can be found.

Possessive quantifiers are a way to prevent the regex engine from trying all permutations. This is primarily useful for performance reasons. You can also use possessive quantifiers to eliminate certain matches.

## How Possessive Quantifiers Work

Several modern regular expression flavors, including the JGsoft, Java and PCRE have a third kind of quantifier: the possessive quantifier. Like a greedy quantifier, a possessive quantifier will repeat the token as many times as possible. Unlike a greedy quantifier, it will *not* give up matches as the engine backtracks. With a possessive quantifier, the deal is all or nothing. You can make a quantifier possessive by placing an extra `+` after it. E.g. «`*`» is greedy, «`*?`» is lazy, and «`*+`» is possessive. «`++`», «`?+`» and «`{n,m}+`» are all possessive as well.

Let's see what happens if we try to match «`"[^"]*+"`» against "`"abc"`". The «"» matches the „"". «`[^"]`» matches „a", „b" and „c" as it is repeated by the star. The final «"» then matches the final „"" and we found an overall match. In this case, the end result is the same, whether we use a greedy or possessive quantifier. There is a slight performance increase though, because the possessive quantifier doesn't have to remember any backtracking positions.

The performance increase can be significant in situations where the regex fails. If the subject is "`"abc`" (no closing quote), the above matching process will happen in the same way, except that the second «"» fails. When using a possessive quantifier, there are no steps to backtrack to. The regular expression does not have any alternation or non-possessive quantifiers that can give up part of their match to try a different permutation of the regular expression. So the match attempt fails immediately when the second «"» fails.

Had we used a greedy quantifier instead, the engine would have backtracked. After the «"» failed at the end of the string, the «`[^"]*`» would give up one match, leaving it with „ab". The «"» would then fail to match "c". «`[^"]*`» backtracks to just „a", and «"» fails to match "b". Finally, «`[^"]*`» backtracks to match zero characters, and «"» fails "a". Only at this point have all backtracking positions been exhausted, and does the engine give up the match attempt. Essentially, this regex performs as many needless steps as there are characters following the unmatched opening quote.

## When Possessive Quantifiers Matter

The main practical benefit of possessive quantifiers is to speed up your regular expression. In particular, possessive quantifiers allow your regex to fail faster. In the above example, when the closing quote fails to match, we *know* the regular expression couldn't have possibly skipped over a quote. So there's no need to backtrack and check for the quote. We make the regex engine aware of this by making the quantifier possessive. In fact, some engines, including the JGsoft engine detect that «`[^"]*`» and «"» are mutually exclusive when compiling your regular expression, and automatically make the star possessive.

Now, linear backtracking like a regex with a single quantifier does is pretty fast. It's unlikely you'll notice the speed difference. However, when you're nesting quantifiers, a possessive quantifier may save your day. Nesting quantifiers means that you have one or more repeated tokens inside a group, and the group is also repeated. That's when catastrophic backtracking often rears its ugly head. In such cases, you'll depend on possessive quantifiers and/or atomic grouping to save the day.

## Possessive Quantifiers Can Change The Match Result

Using possessive quantifiers can change the result of a match attempt. Since no backtracking is done, and matches that would require a greedy quantifier to backtrack will not be found with a possessive quantifier. E.g. «`".*"`» will match „`"abc"`" in "`"abc"x`", but «`".*+"`» will not match this string at all.

In both regular expressions, the first «"» will match the first „"" in the string. The repeated dot then matches the remainder of the string „`abc"x`". The second «"» then fails to match at the end of the string.

Now, the paths of the two regular expressions diverge. The possessive dot-star wants it all. No backtracking is done. Since the «"» failed, there are no permutations left to try, and the overall match attempt fails. The greedy dot-star, while initially grabbing everything, is willing to give back. It will backtrack one character at a time. Backtracking to „`abc"`", «"» fails to match "`x`". Backtracking to „`abc`", «"» matches „"". An overall match „`"abc"`" was found.

Essentially, the lesson here is that when using possessive quantifiers, you need to make sure that whatever you're applying the possessive quantifier to should not be able to match what should follow it. The problem in the above example is that the dot also matches the closing quote. This prevents us from using a possessive quantifier. The negated character class in the previous section cannot match the closing quote, so we can make it possessive.

## Using Atomic Grouping Instead of Possessive Quantifiers

Technically, possessive quantifiers are a notational convenience to place an atomic group around a single quantifier. All regex flavors that support possessive quantifiers also support atomic grouping. But not all regex flavors that support atomic grouping support possessive quantifiers. With those flavors, you can achieve the exact same results using an atomic group.

Basically, instead of «`X*+`», write «`(?>X*)`». It is important to notice that both the quantified token X and the quantifier are inside the atomic group. Even if X is a group, you still need to put an extra atomic group around it to achieve the same effect. «`(?:a|b)*+`» is equivalent to «`(?>(?:a|b)*)`» but not to «`(?>a|b)*`».

The latter is a valid regular expression, but it won't have the same effect when used as part of a larger regular expression.

E.g. «(?:a|b)*+b» and «(?>(?:a|b)*)b» both fail to match "b". «a|b» will match the „b". The star is satisfied, and the fact that it's possessive or the atomic group will cause the star to forget all its backtracking positions. The second «b» in the regex has nothing left to match, and the overall match attempt fails.

In the regex «(?>a|b)*b», the atomic group forces the alternation to give up its backtracking positions. I.e. if an „a" is matched, it won't come back to try «b» if the rest of the regex fails. Since the star is outside of the group, it is a normal, greedy star. When the second «b» fails, the greedy star will backtrack to zero iterations. Then, the second «b» matches the „b" in the subject string.

This distinction is particularly important when converting a regular expression written by somebody else using possessive quantifiers to a regex flavor that doesn't have possessive quantifiers. You could, of course, let a tool like RegexBuddy do the job for you.

# 16. Atomic Grouping

An atomic group is a group that, when the regex engine exits from it, automatically throws away all backtracking positions remembered by any tokens inside the group. Atomic groups are non-capturing. The syntax is «`(?>group)`». Lookaround groups are also atomic. Atomic grouping is supported by most modern regular expression flavors, including the JGsoft flavor, Java, PCRE, .NET, Perl and Ruby. The first three of these also support possessive quantifiers, which are essentially a notational convenience for atomic grouping.

An example will make the behavior of atomic groups clear. The regular expression «`a(bc|b)c`» (capturing group) matches „`abcc`" and „`abc`". The regex «`a(?>bc|b)c`» (atomic group) matches „`abcc`" but not "`abc`".

When applied to "`abc`", both regexes will match «`a`» to „`a`", «`bc`» to „`bc`", and then «`c`» will fail to match at the end of the string. Here their paths diverge. The regex with the capturing group has remembered a backtracking position for the alternation. The group will give up its match, «`b`» then matches „`b`" and «`c`» matches „`c`". Match found!

The regex with the atomic group, however, exited from an atomic group after «`bc`» was matched. At that point, all backtracking positions for tokens inside the group are discarded. In this example, the alternation's option to try «`b`» at the second position in the string is discarded. As a result, when «`c`» fails, the regex engine has no alternatives left to try.

Of course, the above example isn't very useful. But it does illustrate very clearly how atomic grouping eliminates certain matches. Or more importantly, it eliminates certain match attempts.

## Regex Optimization Using Atomic Grouping

Consider the regex «`\b(integer|insert|in)\b`» and the subject "`integers`". Obviously, because of the word boundaries, these don't match. What's not so obvious is that the regex engine will spend quite some effort figuring this out.

«`\b`» matches at the start of the string, and «`integer`» matches „`integer`". The regex engine makes note that there are two more alternatives in the group, and continues with «`\b`». This fails to match between the "`r`" and "`s`". So the engine backtracks to try the second alternative inside the group. The second alternative matches „`in`", but then fails to match «`s`». So the engine backtracks once more to the third alternative. «`in`» matches „`in`". «`\b`» fails between the "`n`" and "`t`" this time. The regex engine has no more remembered backtracking positions, so it declares failure.

This is quite a lot of work to figure out "`integers`" isn't in our list of words. We can optimize this by telling the regular expression engine that if it can't match «`\b`» after it matched „`integer`", then it shouldn't bother trying any of the other words. The word we've encountered in the subject string is a longer word, and it isn't in our list.

We can do this by turning the capturing group into an atomic group: «`\b(?>integer|insert|in)\b`». Now, when «`integer`» matches, the engine exits from an atomic group, and throws away the backtracking positions it stored for the alternation. When «`\b`» fails, the engine gives up immediately. This savings can be significant when scanning a large file for a long list of keywords. This savings will be vital when your alternatives contain repeated tokens (not to mention repeated groups) that lead to catastrophic backtracking.

Don't be too quick to make all your groups atomic. As we saw in the first example above, atomic grouping can exclude valid matches too. Compare how «`\b(?>integer|insert|in)\b`» and «`\b(?>in|integer|insert)\b`» behave when applied to "`insert`". The former regex matches, while the latter fails. If the groups weren't atomic, both regexes would match. Remember that alternation tries its alternatives from left to right. If the second regex matches „`in`", it won't try the two other alternatives due to the atomic group.

# 17. Lookahead and Lookbehind Zero-Width Assertions

Perl 5 introduced two very powerful constructs: "lookahead" and "lookbehind". Collectively, these are called "lookaround". They are also called "zero-width assertions". They are zero-width just like the start and end of line, and start and end of word anchors that I already explained. The difference is that lookarounds will actually match characters, but then give up the match and only return the result: match or no match. That is why they are called "assertions". They do not consume characters in the string, but only assert whether a match is possible or not. Lookarounds allow you to create regular expressions that are impossible to create without them, or that would get very longwinded without them.

## Positive and Negative Lookahead

Negative lookahead is indispensable if you want to match something not followed by something else. When explaining character classes, I already explained why you cannot use a negated character class to match a "q" not followed by a "u". Negative lookahead provides the solution: «q(?!u)». The negative lookahead construct is the pair of round brackets, with the opening bracket followed by a question mark and an exclamation point. Inside the lookahead, we have the trivial regex «u».

Positive lookahead works just the same. «q(?=u)» matches a q that is followed by a u, without making the u part of the match. The positive lookahead construct is a pair of round brackets, with the opening bracket followed by a question mark and an equals sign.

You can use any regular expression inside the lookahead. (Note that this is not the case with lookbehind. I will explain why below.) Any valid regular expression can be used inside the lookahead. If it contains capturing parentheses, the backreferences will be saved. Note that the lookahead itself does not create a backreference. So it is not included in the count towards numbering the backreferences. If you want to store the match of the regex inside a backreference, you have to put capturing parentheses around the regex inside the lookahead, like this: «(?=(regex))». The other way around will not work, because the lookahead will already have discarded the regex match by the time the backreference is to be saved.

## Regex Engine Internals

First, let's see how the engine applies «q(?!u)» to the string "Iraq". The first token in the regex is the literal «q». As we already know, this will cause the engine to traverse the string until the „q" in the string is matched. The position in the string is now the void behind the string. The next token is the lookahead. The engine takes note that it is inside a lookahead construct now, and begins matching the regex inside the lookahead. So the next token is «u». This does not match the void behind the string. The engine notes that the regex inside the lookahead failed. Because the lookahead is negative, this means that the lookahead has successfully matched at the current position. At this point, the entire regex has matched, and „q" is returned as the match.

Let's try applying the same regex to "quit". «q» matches „q". The next token is the «u» inside the lookahead. The next character is the "u". These match. The engine advances to the next character: "i". However, it is done with the regex inside the lookahead. The engine notes success, and discards the regex match. This causes the engine to step back in the string to "u".

Because the lookahead is negative, the successful match inside it causes the lookahead to fail. Since there are no other permutations of this regex, the engine has to start again at the beginning. Since «q» cannot match anywhere else, the engine reports failure.

Let's take one more look inside, to make sure you understand the implications of the lookahead. Let's apply «q(?=u)i» to "quit". I have made the lookahead positive, and put a token after it. Again, «q» matches „q" and «u» matches „u". Again, the match from the lookahead must be discarded, so the engine steps back from "i" in the string to "u". The lookahead was successful, so the engine continues with «i». But «i» cannot match "u". So this match attempt fails. All remaining attempts will fail as well, because there are no more q's in the string.

## Positive and Negative Lookbehind

Lookbehind has the same effect, but works backwards. It tells the regex engine to temporarily step backwards in the string, to check if the text inside the lookbehind can be matched there. «(?<!a)b» matches a "b" that is not preceded by an "a", using negative lookbehind. It will not match "cab", but will match the „b" (and only the „b") in "bed" or "debt". «(?<=a)b» (positive lookbehind) matches the „b" (and only the „b") in „cab", but does not match "bed" or "debt".

The construct for positive lookbehind is «(?<=text)»: a pair of round brackets, with the opening bracket followed by a question mark, "less than" symbol and an equals sign. Negative lookbehind is written as «(?<!text)», using an exclamation point instead of an equals sign.

## More Regex Engine Internals

Let's apply «(?<=a)b» to "thingamabob". The engine starts with the lookbehind and the first character in the string. In this case, the lookbehind tells the engine to step back one character, and see if an "a" can be matched there. The engine cannot step back one character because there are no characters before the "t". So the lookbehind fails, and the engine starts again at the next character, the "h". (Note that a negative lookbehind would have succeeded here.) Again, the engine temporarily steps back one character to check if an "a" can be found there. It finds a "t", so the positive lookbehind fails again.

The lookbehind continues to fail until the regex reaches the "m" in the string. The engine again steps back one character, and notices that the „a" can be matched there. The positive lookbehind matches. Because it is zero-width, the current position in the string remains at the "m". The next token is «b», which cannot match here. The next character is the second "a" in the string. The engine steps back, and finds out that the "m" does not match «a».

The next character is the first "b" in the string. The engine steps back and finds out that „a" satisfies the lookbehind. «b» matches „b", and the entire regex has been matched successfully. It matches one character: the first „b" in the string.

## Important Notes About Lookbehind

The good news is that you can use lookbehind anywhere in the regex, not only at the start. If you want to find a word not ending with an "s", you could use «\b\w+(?<!s)\b». This is definitely not the same as

«\b\w+[^s]\b». When applied to "John's", the former will match „John" and the latter „John'" (including the apostrophe). I will leave it up to you to figure out why. (Hint: «\b» matches between the apostrophe and the "s"). The latter will also not match single-letter words like "a" or "I". The correct regex without using lookbehind is «\b\w*[^s\W]\b» (star instead of plus, and \W in the character class). Personally, I find the lookbehind easier to understand. The last regex, which works correctly, has a double negation (the \W in the negated character class). Double negations tend to be confusing to humans. Not to regex engines, though.

The bad news is that most regex flavors do not allow you to use just any regex inside a lookbehind, because they cannot apply a regular expression backwards. Therefore, the regular expression engine needs to be able to figure out how many steps to step back before checking the lookbehind.

Therefore, many regex flavors, including those used by Perl and Python, only allow fixed-length strings. You can use any regex of which the length of the match can be predetermined. This means you can use literal text and character classes. You cannot use repetition or optional items. You can use alternation, but only if all options in the alternation have the same length.

PCRE is not fully Perl-compatible when it comes to lookbehind. While Perl requires alternatives inside lookbehind to have the same length, PCRE allows alternatives of variable length. Each alternative still has to be fixed-length.

Java takes things a step further by allowing finite repetition. You still cannot use the star or plus, but you can use the question mark and the curly braces with the max parameter specified. Java recognizes the fact that finite repetition can be rewritten as an alternation of strings with different, but fixed lengths. Unfortunately, the JDK 1.4 and 1.5 have some bugs when you use alternation inside lookbehind. These were fixed in JDK 1.6.

The only regex engines that allow you to use a full regular expression inside lookbehind, including infinite repetition, are the JGsoft engine and the .NET framework RegEx classes.

Finally, flavors like JavaScript, Ruby and Tcl do not support lookbehind at all, even though they do support lookahead.

## Lookaround Is Atomic

The fact that lookaround is zero-width automatically makes it atomic. As soon as the lookaround condition is satisfied, the regex engine forgets about everything inside the lookaround. It will not backtrack inside the lookaround to try different permutations.

The only situation in which this makes any difference is when you use capturing groups inside the lookaround. Since the regex engine does not backtrack into the lookaround, it will not try different permutations of the capturing groups.

For this reason, the regex «(?=(\d+))\w+\1» will never match "123x12". First the lookaround captures „123" into «\1». «\w+» then matches the whole string and backtracks until it matches only „1". Finally, «\w+» fails since «\1» cannot be matched at any position. Now, the regex engine has nothing to backtrack to, and the overall regex fails. The backtracking steps created by «\d+» have been discarded. It never gets to the point where the lookahead captures only "12".

Obviously, the regex engine does try further positions in the string. If we change the subject string, the regex «`(?=(\d+))\w+\1`» will match „`56x56`" in "`456x56`".

If you don't use capturing groups inside lookaround, then all this doesn't matter. Either the lookaround condition can be satisfied or it cannot be. In how many ways it can be satisfied is irrelevant.

# 18. Testing The Same Part of a String for More Than One Requirement

Lookaround, which I introduced in detail in the previous topic, is a very powerful concept. Unfortunately, it is often underused by people new to regular expressions, because lookaround is a bit confusing. The confusing part is that the lookaround is zero-width. So if you have a regex in which a lookahead is followed by another piece of regex, or a lookbehind is preceded by another piece of regex, then the regex will traverse part of the string twice.

To make this clear, I would like to give you another, a bit more practical example. Let's say we want to find a word that is six letters long and contains the three consecutive letters "`cat`". Actually, we can match this without lookaround. We just specify all the options and lump them together using alternation: «`cat\w{3}|\wcat\w{2}|\w{2}cat\w|\w{3}cat`». Easy enough. But this method gets unwieldy if you want to find any word between 6 and 12 letters long containing either "cat", "dog" or "mouse".

## Lookaround to The Rescue

In this example, we basically have two requirements for a successful match. First, we want a word that is 6 letters long. Second, the word we found must contain the word "cat".

Matching a 6-letter word is easy with «`\b\w{6}\b`». Matching a word containing "cat" is equally easy: «`\b\w*cat\w*\b`».

Combining the two, we get: «`(?=\b\w{6}\b)\b\w*cat\w*\b`» . Easy! Here's how this works. At each character position in the string where the regex is attempted, the engine will first attempt the regex inside the positive lookahead. This sub-regex, and therefore the lookahead, matches only when the current character position in the string is at the start of a 6-letter word in the string. If not, the lookahead will fail, and the engine will continue trying the regex from the start at the next character position in the string.

The lookahead is zero-width. So when the regex inside the lookahead has found the 6-letter word, the current position in the string is still at the beginning of the 6-letter word. At this position will the regex engine attempt the remainder of the regex. Because we already know that a 6-letter word can be matched at the current position, we know that «`\b`» matches and that the first «`\w*`» will match 6 times. The engine will then backtrack, reducing the number of characters matched by «`\w*`», until «`cat`» can be matched. If «`cat`» cannot be matched, the engine has no other choice but to restart at the beginning of the regex, at the next character position in the string. This is at the second letter in the 6-letter word we just found, where the lookahead will fail, causing the engine to advance character by character until the next 6-letter word.

If «`cat`» can be successfully matched, the second «`\w*`» will consume the remaining letters, if any, in the 6-letter word. After that, the last «`\b`» in the regex is guaranteed to match where the second «`\b`» inside the lookahead matched. Our double-requirement-regex has matched successfully.

## Optimizing Our Solution

While the above regex works just fine, it is not the most optimal solution. This is not a problem if you are just doing a search in a text editor. But optimizing things is a good idea if this regex will be used repeatedly and/or on large chunks of data in an application you are developing.

You can discover these optimizations by yourself if you carefully examine the regex and follow how the regex engine applies it, as I did above. I said the third and last «`\b`» are guaranteed to match. Since it is zero-width, and therefore does not change the result returned by the regex engine, we can remove them, leaving: «`(?=\b\w{6}\b)\w*cat\w*`». Though the last «`\w*`» is also guaranteed to match, we cannot remove it because it adds characters to the regex match. Remember that the lookahead discards its match, so it does not contribute to the match returned by the regex engine. If we omitted the «`\w*`», the resulting match would be the start of a 6-letter word containing "cat", up to and including "cat", instead of the entire word.

But we can optimize the first «`\w*`». As it stands, it will match 6 letters and then backtrack. But we know that in a successful match, there can never be more than 3 letters before "cat". So we can optimize this to «`\w{0,3}`». Note that making the asterisk lazy would not have optimized this sufficiently. The lazy asterisk would find a successful match sooner, but if a 6-letter word does not contain "cat", it would still cause the regex engine to try matching "cat" at the last two letters, at the last single letter, and even at one character beyond the 6-letter word.

So we have «`(?=\b\w{6}\b)\w{0,3}cat\w*`». One last, minor, optimization involves the first «`\b`». Since it is zero-width itself, there's no need to put it inside the lookahead. So the final regex is: «`\b(?=\w{6}\b)\w{0,3}cat\w*`» .

You could replace the final «`\w*`» with «`\w{0,3}`» too. But it wouldn't make any difference. The lookahead has already checked that we're at a 6-letter word, and «`\w{0,3}cat`» has already matched 3 to 6 letters of that word. Whether we end the regex with «`\w*`» or «`\w{0,3}`» doesn't matter, because either way, we'll be matching all the remaining word characters. Because the resulting match and the speed at which it is found are the same, we may just as well use the version that is easier to type.

## A More Complex Problem

So, what would you use to find any word between 6 and 12 letters long containing either "cat", "dog" or "mouse"? Again we have two requirements, which we can easily combine using a lookahead: « `\b(?=\w{6,12}\b)\w{0,9}(cat|dog|mouse)\w*`» . Very easy, once you get the hang of it. This regex will also put "cat", "dog" or "mouse" into the first backreference.

# 19. Continuing at The End of The Previous Match

The anchor «\G» matches at the position where the previous match ended. During the first match attempt, «\G» matches at the start of the string in the way «\A» does.

Applying «\G\w» to the string "test string" matches „t". Applying it again matches „e". The 3rd attempt yields „s" and the 4th attempt matches the second „t" in the string. The fifth attempt fails. During the fifth attempt, the only place in the string where «\G» matches is after the second t. But that position is not followed by a word character, so the match fails.

## End of The Previous Match vs. Start of The Match Attempt

With some regex flavors or tools, «\G» matches at the start of the match attempt, rather than at the end of the previous match result. This is the case with EditPad Pro, where «\G» matches at the position of the text cursor, rather than the end of the previous match. When a match is found, EditPad Pro will select the match, and move the text cursor to the end of the match. The result is that «\G» matches at the end of the previous match result only when you do not move the text cursor between two searches. All in all, this makes a lot of sense in the context of a text editor.

## \G Magic with Perl

In Perl, the position where the last match ended is a "magical" value that is remembered separately for each string variable. The position is not associated with any regular expression. This means that you can use «\G» to make a regex continue in a subject string where another regex left off.

If a match attempt fails, the stored position for «\G» is reset to the start of the string. To avoid this, specify the continuation modifier /c.

All this is very useful to make several regular expressions work together. E.g. you could parse an HTML file in the following fashion:

```
while ($string =~ m/</g) {
  if ($string =~ m/\GB>/c) {
    # Bold
  } elsif ($string =~ m/\GI>/c) {
    # Italics
  } else {
    # ...etc...
  }
}
```

The regex in the while loop searches for the tag's opening bracket, and the regexes inside the loop check which tag we found. This way you can parse the tags in the file in the order they appear in the file, without having to write a single big regex that matches all tags you are interested in.

## \G in Other Programming Languages

This flexibility is not available with most other programming languages. E.g. in Java, the position for «\G» is remembered by the Matcher object. The Matcher is strictly associated with a single regular expression and a single subject string. What you can do though is to add a line of code to make the match attempt of the second Matcher start where the match of the first Matcher ended. «\G» will then match at this position.

The «\G» token is supported by the JGsoft engine, .NET, Java, Perl and PCRE.

# 20. If-Then-Else Conditionals in Regular Expressions

A special construct «`(?ifthen|else)`» allows you to create conditional regular expressions. If the *if* part evaluates to true, then the regex engine will attempt to match the *then* part. Otherwise, the *else* part is attempted instead. The syntax consists of a pair of round brackets. The opening bracket must be followed by a question mark, immediately followed by the *if* part, immediately followed by the *then* part. This part can be followed by a vertical bar and the *else* part. You may omit the *else* part, and the vertical bar with it.

For the *if* part, you can use the lookahead and lookbehind constructs. Using positive lookahead, the syntax becomes «`(?(?=regex)then|else)`». Because the lookahead has its own parentheses, the *if* and *then* parts are clearly separated.

Remember that the lookaround constructs do not consume any characters. If you use a lookahead as the *if* part, then the regex engine will attempt to match the *then* or *else part* (depending on the outcome of the lookahead) at the same position where the *if* was attempted.

Alternatively, you can check in the *if* part whether a capturing group has taken part in the match thus far. Place the number of the capturing group inside round brackets, and use that as the if part. Note that although the syntax for a conditional check on a backreference is the same as a number inside a capturing group, no capturing group is created. The number and the brackets are part of the if-then-else syntax started with «`(?`».

For the *then* and *else*, you can use any regular expression. If you want to use alternation, you will have to group the *then* or *else* together using parentheses, like in «`(?(?=condition)(then1|then2|then3)|(else1|else2|else3))`». Otherwise, there is no need to use parentheses around the *then* and *else* parts.

## Looking Inside the Regex Engine

The regex «`(a)?b(?(1)c|d)`» matches „`bd`" and „`abc`". It does not match "`bc`", but does match „`bd`" in "`abd`". Let's see how this regular expression works on each of these four subject strings.

When applied to "`bd`", «`a`» fails to match. Since the capturing group containing «`a`» is optional, the engine continues with «`b`» at the start of the subject string. Since the whole group was optional, the group did not take part in the match. Any subsequent backreference to it like «`\1`» will fail. Note that «`(a)?`» is very different from «`(a?)`». In the former regex, the capturing group does not take part in the match if «`a`» fails, and backreferences to the group will fail. In the latter group, the capturing group always takes part in the match, capturing either „`a`" or nothing. Backreferences to a capturing group that took part in the match and captured nothing always succeed. Conditionals evaluating such groups execute the "then" part. In short: if you want to use a reference to a group in a conditional, use «`(a)?`» instead of «`(a?)`».

Continuing with our regex, «`b`» matches „`b`". The regex engine now evaluates the conditional. The first capturing group did not take part in the match at all, so the "else" part or «`d`» is attempted. «`d`» matches „`d`" and an overall match is found.

Moving on to our second subject string "`abc`", «`a`» matches „`a`", which is captured by the capturing group. Subsequently, «`b`» matches „`b`". The regex engine again evaluates the conditional. The capturing group took part in the match, so the "then" part or «`c`» is attempted. «`c`» matches „`c`" and an overall match is found.

Our third subject "bc" does not start with "a", so the capturing group does not take part in the match attempt, like we saw with the first subject string. «b» still matches „b", and the engine moves on to the conditional. The first capturing group did not take part in the match at all, so the "else" part or «d» is attempted. «d» does not match "c" and the match attempt at the start of the string fails. The engine does try again starting at the second character in the string, but fails since «b» does not match "c".

The fourth subject "abd" is the most interesting one. Like in the second string, the capturing group grabs the „a" and the «b» matches. The capturing group took part in the match, so the "then" part or «c» is attempted. «c» fails to match "d", and the match attempt fails. Note that the "else" part is not attempted at this point. The capturing group took part in the match, so only the "then" part is used. However, the regex engine isn't done yet. It will restart the regular expression from the beginning, moving ahead one character in the subject string.

Starting at the second character in the string, «a» fails to match "b". The capturing group does not take part in the second match attempt which started at the second character in the string. The regex engine moves beyond the optional group, and attempts «b», which matches. The regex engine now arrives at the conditional in the regex, and at the third character in the subject string. The first capturing group did not take part in the current match attempt, so the "else" part or «d» is attempted. «d» matches „d" and an overall match „bd" is found.

If you want to avoid this last match result, you need to use anchors. «^(a)?b(?(1)c|d)$» does not find any matches in the last subject string. The caret will fail to match at the second and third characters in the string.

## Regex Flavors

Conditionals are supported by the JGsoft engine, Perl, PCRE and the .NET framework. All these flavors, except Perl, also support named capturing groups. They allow you to use the name of a capturing group instead of its number as the *if* test, e.g.: «(?<test>a)?b(?(test)c|d)».

Python supports conditionals using a numbered or named capturing group. Python does not support conditionals using lookaround, even though Python does support lookaround outside conditionals. Instead of a conditional like «(?(?=regex)then|else)», you can alternate two opposite lookarounds: «(?=regex)then|(?!regex)else)».

## Example: Extract Email Headers

The regex «^((From|To)|Subject): ((?(2)\w+@\w+\.[a-z]+|.+))» extracts the From, To, and Subject headers from an email message. The name of the header is captured into the first backreference. If the header is the From or To header, it is captured into the second backreference as well.

The second part of the pattern is the if-then-else conditional «(?(2)\w+@\w+\.[a-z]+|.+))». The *if* part checks whether the second capturing group took part in the match thus far. It will have taken part if the header is the From or To header. In that case, the *then* part of the conditional «\w+@\w+\.[a-z]+» tries to match an email address. To keep the example simple, we use an overly simple regex to match the email address, and we don't try to match the display name that is usually also part of the From or To header.

If the second capturing group did not participate in the match this far, the *else* part «.+» is attempted instead. This simply matches the remainder of the line, allowing for any test subject.

Finally, we place an extra pair of round brackets around the conditional. This captures the contents of the email header matched by the conditional into the third backreference. The conditional itself does not capture anything. When implementing this regular expression, the first capturing group will store the name of the header ("From", "To", or "Subject"), and the third capturing group will store the value of the header.

You could try to match even more headers by putting another conditional into the "else" part. E.g. «`^((From|To)|(Date)|Subject): ((?(2)\w+@\w+\.[a-z]+|(?(3)mm/dd/yyyy|.+))`» would match a "From", "To", "Date" or "Subject", and use the regex «`mm/dd/yyyy`» to check whether the date is valid. Obviously, the date validation regex is just a dummy to keep the example simple. The header is captured in the first group, and its validated contents in the fourth group.

As you can see, regular expressions using conditionals quickly become unwieldy. I recommend that you only use them if one regular expression is all your tool allows you to use. When programming, you're far better of using the regex «`^(From|To|Date|Subject): (.+)`» to capture one header with its unvalidated contents. In your source code, check the name of the header returned in the first capturing group, and then use a second regular expression to validate the contents of the header returned in the second capturing group of the first regex. Though you'll have to write a few lines of extra code, this code will be much easier to understand and maintain. If you precompile all the regular expressions, using multiple regular expressions will be just as fast, if not faster, than the one big regex stuffed with conditionals.

# 21. XML Schema Character Classes

XML Schema Regular Expressions support the usual six shorthand character classes, plus four more. These four aren't supported by any other regular expression flavor. «`\i`» matches any character that may be the first character of an XML name, i.e. «`[_:A-Za-z]`». «`\c`» matches any character that may occur after the first character in an XML name, i.e. «`[-._:A-Za-z0-9]`». «`\I`» and «`\C`» are the respective negated shorthands. Note that the «`\c`» shorthand syntax conflicts with the control character syntax used in many other regex flavors.

You can use these four shorthands both inside and outside character classes using the bracket notation. They're very useful for validating XML references and values in your XML schemas. The regular expression «`\i\c*`» matches an XML name like „`xml:schema`". In other regular expression flavors, you'd have to spell this out as «`[_:A-Za-z][-._:A-Za-z0-9]*`». The latter regex also works with XML's regular expression flavor. It just takes more time to type in.

The regex «`<\i\c*\s*>`» matches an opening XML tag without any attributes. «`</\i\c*\s*>`» matches any closing tag. «`<\i\c*(\s+\i\c*\s*=\s*("[^"]*"|'[^']*'))*\s*>`» matches an opening tag with any number of attributes. Putting it all together, «`<(\i\c*(\s+\i\c*\s*=\s*("[^"]*"|'[^']*'))*|/\i\c*)\s*>`» matches either an opening tag with attributes or a closing tag.

## Character Class Subtraction

While the regex flavor it defines is quite limited, the XML Schema adds a new regular expression feature not previously seen in any (popular) regular expression flavor: character class subtraction. Currently, this feature is only supported by the JGsoft and .NET regex engines (in addition to those implementing the XML Schema standard).

Character class subtraction makes it easy to match any single character present in one list (the character class), but not present in another list (the subtracted class). The syntax for this is `[class-[subtract]]`. If the character after a hyphen is an opening bracket, XML regular expressions interpret the hyphen as the subtraction operator rather than the range operator. E.g. «`[a-z-[aeiuo]]`» matches a single letter that is not a vowel (i.e. a single consonant). Without the character class subtraction feature, the only way to do this would be to list all consonants: «`[b-df-hj-np-tv-z]`».

This feature is more than just a notational convenience, though. You can use the full character class syntax within the subtracted character class. E.g. to match all Unicode letters except ASCII letters (i.e. all non-English letters), you could easily use «`[\p{L}-[\p{IsBasicLatin}]]`».

## Nested Character Class Subtraction

Since you can use the full character class syntax within the subtracted character class, you can subtract a class from the class being subtracted. E.g. «`[0-9-[0-6-[0-3]]]`» first subtracts `0-3` from `0-6`, yielding «`[0-9-[4-6]]`», or «`[0-37-9]`», which matches any character in the string "`0123789`".

The class subtraction must always be the last element in the character class. `[0-9-[4-6]a-f]` is not a valid regular expression. It should be rewritten as «`[0-9a-f-[4-6]]`». The subtraction works on the whole class.

E.g. «`[\p{Ll}\p{Lu}-[\p{IsBasicLatin}]]`» matches all uppercase and lowercase Unicode letters, except any ASCII letters. The `\p{IsBasicLatin}` is subtracted from the combination of `\p{Ll}\p{Lu}` rather than from `\p{Lu}` alone. This regex will not match "`abc`".

While you can use nested character class subtraction, you cannot subtract two classes sequentially. To subtract ASCII letters and Greek letters from a class with all Unicode letters, combine the ASCII and Greek letters into one class, and subtract that, as in «`[\p{L}-[\p{IsBasicLatin}\p{IsGreek}]]`».

## Notational Compatibility with Other Regex Flavors

Note that a regex like «`[a-z-[aeiuo]]`» will not cause any errors in regex flavors that do not support character class subtraction. But it won't match what you intended either. E.g. in Perl, this regex consists of a character class followed by a literal «`]`». The character class matches a character that is either in the range a-z, or a hyphen, or an opening bracket, or a vowel. Since the a-z range and the vowels are redundant, you could write this character class as «`[a-z-[]`» or «`[-[a-z]`». A hyphen after a range is treated as a literal character, just like a hyphen immediately after the opening bracket. This is true in all regex flavors, including XML. E.g. «`[a-z-_]`» matches a lowercase letter, a hyphen or an underscore in both Perl and XML Schema.

While the last paragraph strictly speaking means that the XML Schema character class syntax is incompatible with Perl and the majority of other regex flavors, in practice there's no difference. Using non-alphanumeric characters in character class ranges is very bad practice, as it relies on the order of characters in the ASCII character table, which makes the regular expression hard to understand for the programmer who inherits your work. E.g. while «`[A-[]`» would match any upper case letter or an opening square bracket in Perl, this regex is much clearer when written as «`[A-Z[]`». The former regex would cause an error in XML Schema, because it interprets `-[]` as an empty subtracted class, leaving an unbalanced `[`.

# 22. POSIX Bracket Expressions

POSIX bracket expressions are a special kind of character classes. POSIX bracket expressions match one character out of a set of characters, just like regular character classes. They use the same syntax with square brackets. A hyphen creates a range, and a caret at the start negates the bracket expression.

One key syntactic difference is that the backslash is NOT a metacharacter in a POSIX bracket expression. So in POSIX, the regular expression «`[\d]`» matches a „`\`" or a „`d`". To match a „`]`", put it as the first character after the opening `[` or the negating `^`. To match a „`-`", put it right before the closing `]`. To match a „`^`", put it before the final literal `-` or the closing `]`. Put together, «`[]\d^-]`» matches „`]`", „`\`", „`d`", „`^`" or „`-`".

The main purpose of the bracket expressions is that they adapt to the user's or application's locale. A locale is a collection of rules and settings that describe language and cultural conventions, like sort order, date format, etc. The POSIX standard also defines these locales.

Generally, only POSIX-compliant regular expression engines have proper and full support for POSIX bracket expressions. Some non-POSIX regex engines support POSIX character classes, but usually don't support collating sequences and character equivalents. Regular expression engines that support Unicode use Unicode properties and scripts to provide functionality similar to POSIX bracket expressions. In Unicode regex engines, shorthand character classes like «`\w`» normally match all relevant Unicode characters, alleviating the need to use locales.

## Character Classes

Don't confuse the POSIX term "character class" with what is normally called a regular expression character class. «`[x-z0-9]`» is an example of what we call a "character class" and POSIX calls a "bracket expression". `[:digit:]` is a POSIX character class, used inside a bracket expression like «`[x-z[:digit:]]`». These two regular expressions match exactly the same: a single character that is either „`x`", „`y`", „`z`" or a digit. The class names must be written all lowercase.

POSIX bracket expressions can be negated. «`[^x-z[:digit:]]`» matches a single character that is not x, y, z or a digit. A major difference between POSIX bracket expressions and the character classes in other regex flavors is that POSIX bracket expressions treat the backslash as a literal character. This means you can't use backslashes to escape the closing bracket (]), the caret (^) and the hyphen (-). To include a caret, place it anywhere except right after the opening bracket. «`[x^]`» matches an x or a caret. You can put the closing bracket right after the opening bracket, or the negating caret. «`[]x]`» matches a closing bracket or an x. «`[^]x]`» matches any character that is not a closing bracket or an x. The hyphen can be included right after the opening bracket, or right before the closing bracket, or right after the negating caret. Both «`[-x]`» and «`[x-]`» match an x or a hyphen.

Exactly which POSIX character classes are available depends on the POSIX locale. The following are usually supported, often also by regex engines that don't support POSIX itself. I've also indicated equivalent character classes that you can use in ASCII and Unicode regular expressions if the POSIX classes are unavailable. Some classes also have Perl-style shorthand equivalents.

Java does not support POSIX bracket expressions, but does support POSIX character classes using the `\p` operator. Though the `\p` syntax is borrowed from the syntax for Unicode properties, the POSIX classes in Java only match ASCII characters as indicated below. The class names are case sensitive. Unlike the POSIX

syntax which can only be used inside a bracket expression, Java's `\p` can be used inside and outside bracket expressions.

POSIX: «`[:alnum:]`»
Description: Alphanumeric characters
ASCII: «`[a-zA-Z0-9]`»
Unicode: «`[\p{L&}\p{Nd}]`»
Shorthand:
Java: «`\p{Alnum}`»

POSIX: «`[:alpha:]`»
Description: Alphabetic characters
ASCII: «`[a-zA-Z]`»
Unicode: «`\p{L&}`»
Shorthand:
Java: «`\p{Alpha}`»

POSIX: «`[:ascii:]`»
Description: ASCII characters
ASCII: «`[\x00-\x7F]`»
Unicode: «`\p{InBasicLatin}`»
Shorthand:
Java: «`\p{ASCII}`»

POSIX: «`[:blank:]`»
Description: Space and tab
ASCII: «`[ \t]`»
Unicode: «`[\p{Zs}\t]`»
Shorthand:
Java: «`\p{Blank}`»

POSIX: «`[:cntrl:]`»
Description: Control characters
ASCII: «`[\x00-\x1F\x7F]`»
Unicode: «`\p{Cc}`»
Shorthand:
Java: «`\p{Cntrl}`»

POSIX: «`[:digit:]`»
Description: Digits
ASCII: «`[0-9]`»
Unicode: «`\p{Nd}`»
Shorthand: «`\d`»
Java: «`\p{Digit}`»

POSIX: «`[:graph:]`»
Description: Visible characters (i.e. anything except spaces, control characters, etc.)
ASCII: «`[\x21-\x7E]`»
Unicode: «`[^\p{Z}\p{C}]`»
Shorthand:
Java: «`\p{Graph}`»

POSIX: «`[:lower:]`»
Description: Lowercase letters
ASCII: «`[a-z]`»
Unicode: «`\p{Ll}`»
Shorthand:
Java: «`\p{Lower}`»

POSIX: «`[:print:]`»
Description: Visible characters and spaces (i.e. anything except control characters, etc.)
ASCII: «`[\x20-\x7E]`»
Unicode: «`\P{C}`»
Shorthand:
Java: «`\p{Print}`»

POSIX: «`[:punct:]`»
Description: Punctuation and symbols.
ASCII: «`[!"#$%&'()*+,\-./:;<=>?@[\\\]^_`{|}~]`»
Unicode: «`[\p{P}\p{S}]`»
Shorthand:
Java: «`\p{Punct}`»

POSIX: «`[:space:]`»
Description: All whitespace characters, including line breaks
ASCII: «`[ \t\r\n\v\f]`»
Unicode: «`[\p{Z}\t\r\n\v\f]`»
Shorthand: «`\s`»
Java: «`\p{Space}`»

POSIX: «`[:upper:]`»
Description: Uppercase letters
ASCII: «`[A-Z]`»
Unicode: «`\p{Lu}`»
Shorthand:
Java: «`\p{Upper}`»

POSIX: «`[:word:]`»
Description: Word characters (letters, numbers and underscores)
ASCII: «`[A-Za-z0-9_]`»
Unicode: «`[\p{L}\p{N}\p{Pc}]`»
Shorthand: «`\w`»
Java:

POSIX: «`[:xdigit:]`»
Description: Hexadecimal digits
ASCII: «`[A-Fa-f0-9]`»
Unicode: «`[A-Fa-f0-9]`»
Shorthand:
Java: «`\p{XDigit}`»

## Collating Sequences

A POSIX locale can have collating sequences to describe how certain characters or groups of characters should be ordered. E.g. in Spanish, "ll" like in "tortilla" is treated as one character, and is ordered between "l" and "m" in the alphabet. You can use the collating sequence element [.span-ll.] inside a bracket expression to match „ll". E.g. the regex «torti[[.span-ll.]]a» matches „tortilla". Notice the double square brackets. One pair for the bracket expression, and one pair for the collating sequence.

I do not know of any regular expression engine that support collating sequences, other than POSIX-compliant engines part of a POSIX-compliant system.

Note that a fully POSIX-compliant regex engine will treat "ll" as a single character when the locale is set to Spanish. This means that «torti[^x]a» also matches „tortilla". «[^x]» matches a single character that is not an "x", which includes „ll" in the Spanish POSIX locale.

In any other regular expression engine, or in a POSIX engine not using the Spanish locale, «torti[^x]a» will match the misspelled word „tortila" but will not match „tortilla", as «[^x]» cannot match the two characters "ll".

Finally, note that not all regex engines claiming to implement POSIX regular expressions actually have full support for collating sequences. Sometimes, these engines use the regular expression syntax defined by POSIX, but don't have full locale support. You may want to try the above matches to see if the engine you're using does. E.g. Tcl's regexp command supports collating sequences, but Tcl only supports the Unicode locale, which does not define any collating sequences. The result is that in Tcl, a collating sequence specifying a single character will match just that character, and all other collating sequences will result in an error.

## Character Equivalents

A POSIX locale can define character equivalents that indicate that certain characters should be considered as identical for sorting. E.g. in French, accents are ignored when ordering words. "élève" comes before "être" which comes before "événement". "é" and "ê" are all the same as "e", but "l" comes before "t" which comes before "v". With the locale set to French, a POSIX-compliant regular expression engine will match „e", „é", „è" and „ê" when you use the collating sequence [=e=] in the bracket expression «[[=e=]]».

If a character does not have any equivalents, the character equivalence token simply reverts to the character itself. E.g. «[[=x=][=z=]]» is the same as «[xz]» in the French locale.

Like collating sequences, POSIX character equivalents are not available in any regex engine that I know of, other than those following the POSIX standard. And those that do may not have the necessary POSIX locale support. Here too Tcl's regexp command supports character equivalents, but Unicode locale, the only one Tcl supports, does not define any character equivalents. This effectively means that «[[=x=]]» and «[x]» are exactly the same in Tcl, and will only match „x", for any character you may try instead of "x".

# 23. Adding Comments to Regular Expressions

If you have worked through the entire tutorial, I guess you will agree that regular expressions can quickly become rather cryptic. Therefore, many modern regex flavors allow you to insert comments into regexes. The syntax is «`(?#comment)`» where "comment" can be whatever you want, as long as it does not contain a closing round bracket. The regex engine ignores everything after the «`(?#`» until the first closing round bracket.

E.g. I could clarify the regex to match a valid date by writing it as «`(?#year)(19|20)\d\d[-/.](?#month)(0[1-9]|1[012])[- /.](?#day)(0[1-9]|[12][0-9]|3[01])`» . Now it is instantly obvious that this regex matches a date in yyyy-mm-dd format. Some software, such as RegexBuddy, EditPad Pro and PowerGREP can apply syntax coloring to regular expressions while you write them. That makes the comments really stand out, enabling the right comment in the right spot to make a complex regular expression much easier to understand.

Regex comments are supported by the JGsoft engine, .NET, Perl, PCRE, Python and Ruby.

To make your regular expression even more readable, you can turn on free-spacing mode. All flavors that support comments also support free-spacing mode. In addition, Java supports free-spacing mode, even though it doesn't support `(?#)`-style comments.

# 24. Free-Spacing Regular Expressions

The JGsoft engine, .NET, Java, Perl, PCRE, Python, Ruby and XPath support a variant of the regular expression syntax called free-spacing mode. You can turn on this mode with the «`(?x)`» mode modifier, or by turning on the corresponding option in the application or passing it to the regex constructor in your programming language.

In free-spacing mode, whitespace between regular expression tokens is ignored. Whitespace includes spaces, tabs and line breaks. Note that only whitespace *between* tokens is ignored. E.g. «`a b c`» is the same as «`abc`» in free-spacing mode, but «`\ d`» and «`\d`» are not the same. The former matches „ d", while the latter matches a digit. «`\d`» is a single regex token composed of a backslash and a "d". Breaking up the token with a space gives you an escaped space (which matches a space), and a literal "d".

Likewise, grouping modifiers cannot be broken up. «`(?>atomic)`» is the same as «`(?> ato mic )`» and as «`( ?>ato mic)`». They all match the same atomic group. They're not the same as `(? >atomic)`. In fact, the latter will cause a syntax error. The `?>` grouping modifier is a single element in the regex syntax, and must stay together. This is true for all such constructs, including lookaround, named groups, etc.

A character class is also treated as a single token. «`[abc]`» is not the same as «`[ a b c ]`». The former matches one of three letters, while the latter matches those three letters or a space. In other words: free-spacing mode has no effect inside character classes. Spaces and line breaks inside character classes will be included in the character class.

This means that in free-spacing mode, you can use «`\ `» or «`[ ]`» to match a single space. Use whichever you find more readable.

Java, however, does not treat a character class as a single token in free-spacing mode. Java does ignore whitespace and comments inside character classes. So in Java's free-spacing mode, «`[abc]`» is identical to «`[ a b c ]`» and «`\ `» is the only way to match a space. However. even in free-spacing mode, the negating caret must appear immediately after the opening bracket. «`[ ^ a b c ]`» matches any of the four characters „^", „a", „b" or „c" just like «`[abc^]`» would. With the negating caret in the proper place, «`[^ a b c ]`» matches any character that is not "a", "b" or "c".

## Comments in Free-Spacing Mode

Another feature of free-spacing mode is that the # character starts a comment. The comment runs until the end of the line. Everything from the # until the next line break character is ignored.

The XPath flavor does not support comments within the regular expression. The # is always treated as a literal character.

Putting it all together, I could clarify the regex to match a valid date by writing it across multiple lines as:

```
# Match a 20th or 21st century date in yyyy-mm-dd format
(19|20)\d\d                # year (group 1)
[- /.]                     # separator
(0[1-9]|1[012])            # month (group 2)
[- /.]                     # separator
(0[1-9]|[12][0-9]|3[01])   # day (group 3)
```

Part 3

# Regular Expression Examples

# 1. Sample Regular Expressions

Below, you will find many example patterns that you can use for and adapt to your own purposes. Key techniques used in crafting each regex are explained, with links to the corresponding pages in the tutorial where these concepts and techniques are explained in great detail.

If you are new to regular expressions, you can take a look at these examples to see what is possible. Regular expressions are very powerful. They do take some time to learn. But you will earn back that time quickly when using regular expressions to automate searching or editing tasks in EditPad Pro or PowerGREP, or when writing scripts or applications in a variety of languages.

RegexBuddy offers the fastest way to get up to speed with regular expressions. RegexBuddy will analyze any regular expression and present it to you in a clearly to understand, detailed outline. The outline links to RegexBuddy's regex tutorial (the same one you find on this website), where you can always get in-depth information with a single click.

Oh, and you definitely do not need to be a programmer to take advantage of regular expressions!

## Grabbing HTML Tags

«`<TAG\b[^>]*>(.*?)</TAG>`» matches the opening and closing pair of a specific HTML tag. Anything between the tags is captured into the first backreference. The question mark in the regex makes the star lazy, to make sure it stops before the first closing tag rather than before the last, like a greedy star would do. This regex will not properly match tags nested inside themselves, like in "`<TAG>one<TAG>two</TAG>one</TAG>`".

«`<([A-Z][A-Z0-9]*)\b[^>]*>(.*?)</\1>`» will match the opening and closing pair of any HTML tag. Be sure to turn off case sensitivity. The key in this solution is the use of the backreference «`\1`» in the regex. Anything between the tags is captured into the second backreference. This solution will also not match tags nested in themselves.

## Trimming Whitespace

You can easily trim unnecessary whitespace from the start and the end of a string or the lines in a text file by doing a regex search-and-replace. Search for «`^[ \t]+`» and replace with nothing to delete leading whitespace (spaces and tabs). Search for «`[ \t]+$`» to trim trailing whitespace. Do both by combining the regular expressions into «`^[ \t]+|[ \t]+$`» . Instead of `[ \t]` which matches a space or a tab, you can expand the character class into «`[ \t\r\n]`» if you also want to strip line breaks. Or you can use the shorthand «`\s`» instead.

## IP Addresses

Matching an IP address is another good example of a trade-off between regex complexity and exactness. «`\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b`» will match any IP address just fine, but will also match

„`999.999.999.999`" as if it were a valid IP address. Whether this is a problem depends on the files or data you intend to apply the regex to. To restrict all 4 numbers in the IP address to 0..255, you can use this complex beast: «`\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.»«(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.»«(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.»«(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b`» (everything on a single line). The long regex stores each of the 4 numbers of the IP address into a capturing group. You can use these groups to further process the IP number.

If you don't need access to the individual numbers, you can shorten the regex with a quantifier to: «`\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}»«(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b`» . Similarly, you can shorten the quick regex to «`\b(?:\d{1,3}\.){3}\d{1,3}\b`»

## More Detailed Examples

Numeric Ranges. Since regular expressions work with text rather than numbers, matching specific numeric ranges requires a bit of extra care.

Matching a Floating Point Number. Also illustrates the common mistake of making everything in a regular expression optional.

Matching an Email Address. There's a lot of controversy about what is a proper regex to match email addresses. It's a perfect example showing that you need to know exactly what you're trying to match (and what not), and that there's always a trade-off between regex complexity and accuracy.

Matching Valid Dates. A regular expression that matches 31-12-1999 but not 31-13-1999.

Finding or Verifying Credit Card Numbers. Validate credit card numbers entered on your order form. Find credit card numbers in documents for a security audit.

Matching Complete Lines. Shows how to match complete lines in a text file rather than just the part of the line that satisfies a certain requirement. Also shows how to match lines in which a particular regex does *not* match.

Removing Duplicate Lines or Items. Illustrates simple yet clever use of capturing parentheses or backreferences.

Regex Examples for Processing Source Code. How to match common programming language syntax such as comments, strings, numbers, etc.

Two Words Near Each Other. Shows how to use a regular expression to emulate the "near" operator that some tools have.

## Common Pitfalls

Catastrophic Backtracking. If your regular expression seems to take forever, or simply crashes your application, it has likely contracted a case of catastrophic backtracking. The solution is usually to be more

specific about what you want to match, so the number of matches the engine has to try doesn't rise exponentially.

Making Everything Optional. If all the parts in your regex are optional, it will match a zero-width string anywhere. Your regex will need to express the facts that different parts are optional depending on which parts are present.

Repeating a Capturing Group vs. Capturing a Repeated Group. Repeating a capturing group will capture only the last iteration of the group. Capture a repeated group if you want to capture all iterations.

Mixing Unicode and 8-bit Character Codes. Using 8-bit character codes like «`\x80`» with a Unicode engine and subject string may give unexpected results.

# 2. Matching Numeric Ranges with a Regular Expression

Since regular expressions deal with text rather than with numbers, matching a number in a given range takes a little extra care. You can't just write «`[0-255]`» to match a number between 0 and 255. Though a valid regex, it matches something entirely different. «`[0-255]`» is a character class with three elements: the character range 0-2, the character 5 and the character 5 (again). This character class matches a single digit 0, 1, 2 or 5, just like «`[0125]`».

Since regular expressions work with text, a regular expression engine treats "`0`" as a single character, and "`255`" as three characters. To match all characters from 0 to 255, we'll need a regex that matches between one and three characters.

The regex «`[0-9]`» matches single-digit numbers 0 to 9. «`[1-9][0-9]`» matches double-digit numbers 10 to 99. That's the easy part.

Matching the three-digit numbers is a little more complicated, since we need to exclude numbers 256 through 999. «`1[0-9][0-9]`» takes care of 100 to 199. «`2[0-4][0-9]`» matches 200 through 249. Finally, «`25[0-5]`» adds 250 till 255.

As you can see, you need to split up the numeric range in ranges with the same number of digits, and each of those ranges that allow the same variation for each digit. In the 3-digit range in our example, numbers starting with 1 allow all 10 digits for the following two digits, while numbers starting with 2 restrict the digits that are allowed to follow.

Putting this all together using alternation we get: «`[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]`». This matches the numbers we want, with one caveat: regular expression searches usually allow partial matches, so our regex would match „`123`" in "`12345`". There are two solutions to this.

If you're searching for these numbers in a larger document or input string, use word boundaries to require a non-word character (or no character at all) to precede and to follow any valid match. The regex then becomes «`\b([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\b`». Since the alternation operator has the lowest precedence of all, the round brackets are required to group the alternatives together. This way the regex engine will try to match the first word boundary, then try all the alternatives, and then try to match the second word boundary after the numbers it matched. Regular expression engines consider all alphanumeric characters, as well as the underscore, as word characters.

If you're using the regular expression to validate input, you'll probably want to check that the entire input consists of a valid number. To do this, use anchors instead of word boundaries: «`^([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$`».

Here are a few more common ranges that you may want to match:

- 000..255: «`^([01][0-9][0-9]|2[0-4][0-9]|25[0-5])$`»
- 0 or 000..255: «`^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])$`»
- 0 or 000..127: «`^(0?[0-9]?[0-9]|1[0-1][0-9]|12[0-7])$`»
- 0..999: «`^([0-9]|[1-9][0-9]|[1-9][0-9][0-9])$`»
- 000..999: «`^[0-9]{3}$`»
- 0 or 000..999: «`^[0-9]{1,3}$`»
- 1..999: «`^([1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`»

- 001..999: «`^(00[1-9]|0[1-9][0-9]|[1-9][0-9][0-9])$`»
- 1 or 001..999: «`^(0{0,2}[1-9]|0?[1-9][0-9]|[1-9][0-9][0-9])$`»
- 0 or 00..59: «`^[0-5]?[0-9]$`»
- 0 or 000..366: «`^(0?[0-9]?[0-9]|[1-2][0-9][0-9]|3[0-5][0-9]|36[0-6])$`»

# 3. Matching Floating Point Numbers with a Regular Expression

In this example, I will show you how you can avoid a common mistake often made by people inexperienced with regular expressions. As an example, we will try to build a regular expression that can match any floating point number. Our regex should also match integers, and floating point numbers where the integer part is not given (i.e. zero). We will not try to match numbers with an exponent, such as 1.5e8 (150 million in scientific notation).

At first thought, the following regex seems to do the trick: «`[-+]?[0-9]*\.?[0-9]*`». This defines a floating point number as an optional sign, followed by an optional series of digits (integer part), followed by an optional dot, followed by another optional series of digits (fraction part).

Spelling out the regex in words makes it obvious: everything in this regular expression is optional. This regular expression will consider a sign by itself or a dot by itself as a valid floating point number. In fact, it will even consider an empty string as a valid floating point number. This regular expression can cause serious trouble if it is used in a scripting language like Perl or PHP to verify user input.

Not escaping the dot is also a common mistake. A dot that is not escaped will match any character, including a dot. If we had not escaped the dot, "`4.4`" would be considered a floating point number, and "`4X4`" too.

When creating a regular expression, it is more important to consider what it should *not* match, than what it should. The above regex will indeed match a proper floating point number, because the regex engine is greedy. But it will also match many things we do not want, which we have to exclude.

Here is a better attempt: «`[-+]?([0-9]*\.[0-9]+|[0-9]+)`». This regular expression will match an optional sign, that is either followed by zero or more digits followed by a dot and one or more digits (a floating point number with optional integer part), or followed by one or more digits (an integer).

This is a far better definition. Any match will include at least one digit, because there is no way around the «`[0-9]+`» part. We have successfully excluded the matches we do not want: those without digits.

We can optimize this regular expression as: «`[-+]?[0-9]*\.?[0-9]+`» .

If you also want to match numbers with exponents, you can use: «`[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?`» . Notice how I made the entire exponent part optional by grouping it together, rather than making each element in the exponent optional.

Finally, if you want to validate if a particular string holds a floating point number, rather than finding a floating point number within longer text, you'll have to anchor your regex: «`^[-+]?[0-9]*\.?[0-9]+$`» or «`^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$`». You can find additional variations of these regexes in RegexBuddy's library.

# 4. How to Find or Validate an Email Address

The regular expression I receive the most feedback, not to mention "bug" reports on, is the one you'll find right in the tutorial's introduction: «`\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b`» . This regular expression, I claim, matches any email address. Most of the feedback I get refutes that claim by showing one email address that this regex doesn't match. Usually, the "bug" report also includes a suggestion to make the regex "perfect".

As I explain below, my claim only holds true when one accepts my definition of what a valid email address really is, and what it's not. If you want to use a different definition, you'll have to adapt the regex. Matching a valid email address is a perfect example showing that (1) before writing a regex, you have to know exactly what you're trying to match, and what not; and (2) there's often a trade-off between what's exact, and what's practical.

The virtue of my regular expression above is that it matches 99% of the email addresses in use today. All the email address it matches can be handled by 99% of all email software out there. If you're looking for a quick solution, you only need to read the next paragraph. If you want to know all the trade-offs and get plenty of alternatives to choose from, read on.

If you want to use the regular expression above, there's two things you need to understand. First, long regexes make it difficult to nicely format paragraphs. So I didn't include «`a-z`» in any of the three character classes. This regex is intended to be used with your regex engine's "case insensitive" option turned on. (You'd be surprised how many "bug" reports I get about that.) Second, the above regex is delimited with word boundaries, which makes it suitable for extracting email addresses from files or larger blocks of text. If you want to check whether the user typed in a valid email address, replace the word boundaries with start-of-string and end-of-string anchors, like this: «`^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$`» .

The previous paragraph also applies to all following examples. You may need to change word boundaries into start/end-of-string anchors, or vice versa. And you will need to turn on the case insensitive matching option.

## Trade-Offs in Validating Email Addresses

Yes, there are a whole bunch of email addresses that my pet regex doesn't match. The most frequently quoted example are addresses on the `.museum` top level domain, which is longer than the 4 letters my regex allows for the top level domain. I accept this trade-off because the number of people using `.museum` email addresses is extremely low. I've never had a complaint that the order forms or newsletter subscription forms on the JGsoft websites refused a `.museum` address (which they would, since they use the above regex to validate the email address).

To include `.museum`, you could use «`^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}$`». However, then there's another trade-off. This regex will match „`john@mail.office`". It's far more likely that John forgot to type in the `.com` top level domain rather than having just created a new `.office` top level domain without ICANN's permission.

This shows another trade-off: do you want the regex to check if the top level domain exists? My regex doesn't. Any combination of two to four letters will do, which covers all existing and planned top level domains except .museum. But it will match addresses with invalid top-level domains like

„`asdf@asdf.asdf`". By not being overly strict about the top-level domain, I don't have to update the regex each time a new top-level domain is created, whether it's a country code or generic domain.

«`^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.(?:[A-Z]{2}|com|org|net|edu|gov|mil|biz|info|mobi|name|aero|asia|jobs|museum)$`» could be used to allow any two-letter country code top level domain, and only specific generic top level domains. By the time you read this, the list might already be out of date. If you use this regular expression, I recommend you store it in a global constant in your application, so you only have to update it in one place. You could list all country codes in the same manner, even though there are almost 200 of them.

Email addresses can be on servers on a subdomain, e.g. „`john@server.department.company.com`". All of the above regexes will match this email address, because I included a dot in the character class after the @ symbol. However, the above regexes will also match „`john@aol...com`" which is not valid due to the consecutive dots. You can exclude such matches by replacing «`[A-Z0-9.-]+\.`» with «`(?:[A-Z0-9-]+\.)+`» in any of the above regexes. I removed the dot from the character class and instead repeated the character class and the following literal dot. E.g. «`\b[A-Z0-9._%+-]+@(?:[A-Z0-9-]+\.)+[A-Z]{2,4}\b`» will match „`john@server.department.company.com`" but not "`john@aol...com`".

Another trade-off is that my regex only allows English letters, digits and a few special symbols. The main reason is that I don't trust all my email software to be able to handle much else. Even though `John.O'Hara@theoharas.com` is a syntactically valid email address, there's a risk that some software will misinterpret the apostrophe as a delimiting quote. E.g. blindly inserting this email address into a SQL will cause it to fail if strings are delimited with single quotes. And of course, it's been many years already that domain names can include non-English characters. Most software and even domain name registrars, however, still stick to the 37 characters they're used to.

The conclusion is that to decide which regular expression to use, whether you're trying to match an email address or something else that's vaguely defined, you need to start with considering all the trade-offs. How bad is it to match something that's not valid? How bad is it not to match something that is valid? How complex can your regular expression be? How expensive would it be if you had to change the regular expression later? Different answers to these questions will require a different regular expression as the solution. My email regex does what I want, but it may not do what you want.

## Regexes Don't Send Email

Don't go overboard in trying to eliminate invalid email addresses with your regular expression. If you have to accept `.museum` domains, allowing any 6-letter top level domain is often better than spelling out a list of all current domains. The reason is that you don't really know whether an address is valid until you try to send an email to it. And even that might not be enough. Even if the email arrives in a mailbox, that doesn't mean somebody still reads that mailbox.

The same principle applies in many situations. When trying to match a valid date, it's often easier to use a bit of arithmetic to check for leap years, rather than trying to do it in a regex. Use a regular expression to find potential matches or check if the input uses the proper syntax, and do the actual validation on the potential matches returned by the regular expression. Regular expressions are a powerful tool, but they're far from a panacea.

## The Official Standard: RFC 2822

Maybe you're wondering why there's no "official" fool-proof regex to match email addresses. Well, there is an official definition, but it's hardly fool-proof.

The official standard is known as RFC 2822. It describes the syntax that valid email addresses must adhere to. You can (but you shouldn't--read on) implement it with this regular expression:

```
«(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-
\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-
\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-
\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])»
```

This regex has two parts: the part before the @, and the part after the @. There are two alternatives for the part before the @: it can either consist of a series of letters, digits and certain symbols, including one or more dots. However, dots may not appear consecutively or at the start or end of the email address. The other alternative requires the part before the @ to be enclosed in double quotes, allowing any string of ASCII characters between the quotes. Whitespace characters, double quotes and backslashes must be escaped with backslashes.

The part after the @ also has two alternatives. It can either be a fully qualified domain name (e.g. regular-expressions.info), or it can be a literal Internet address between square brackets. The literal Internet address can either be an IP address, or a domain-specific routing address.

The reason you shouldn't use this regex is that it only checks the basic syntax of email addresses. `john@aol.com.nospam` would be considered a valid email address according to RFC 2822. Obviously, this email address won't work, since there's no "nospam" top-level domain. It also doesn't guarantee your email software will be able to handle it. Not all applications support the syntax using double quotes or square brackets. In fact, RFC 2822 itself marks the notation using square brackets as obsolete.

We get a more practical implementation of RFC 2822 if we omit the syntax using double quotes and square brackets. It will still match 99.99% of all email addresses in actual use today.

```
«[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-
z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?»
```

A further change you could make is to allow any two-letter country code top level domain, and only specific generic top level domains. This regex filters dummy email addresses like asdf@adsf.adsf. You will need to update it as new top-level domains are added.

```
«[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-
z0-9-]*[a-z0-9])?\.)+(?:[A-
Z]{2}|com|org|net|edu|gov|mil|biz|info|mobi|name|aero|asia|jobs|museum)\b»
```

So even when following official standards, there are still trade-offs to be made. Don't blindly copy regular expressions from online libraries or discussion forums. Always test them on your own data and with your own applications.

# 5. Matching a Valid Date

«`^(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])$`» matches a date in yyyy-mm-dd format from between 1900-01-01 and 2099-12-31, with a choice of four separators. The anchors make sure the entire variable is a date, and not a piece of text containing a date. The year is matched by «`(19|20)\d\d`». I used alternation to allow the first two digits to be 19 or 20. The round brackets are mandatory. Had I omitted them, the regex engine would go looking for 19 or the remainder of the regular expression, which matches a date between 2000-01-01 and 2099-12-31. Round brackets are the only way to stop the vertical bar from splitting up the entire regular expression into two options.

The month is matched by «`0[1-9]|1[012]`», again enclosed by round brackets to keep the two options together. By using character classes, the first option matches a number between 01 and 09, and the second matches 10, 11 or 12.

The last part of the regex consists of three options. The first matches the numbers 01 through 09, the second 10 through 29, and the third matches 30 or 31.

Smart use of alternation allows us to exclude invalid dates such as 2000-00-00 that could not have been excluded without using alternation. To be really perfectionist, you would have to split up the month into various options to take into account the length of the month. The above regex still matches 2003-02-31, which is not a valid date. Making leading zeros optional could be another enhancement.

If you want to require the delimiters to be consistent, you could use a backreference. «`^(19|20)\d\d([- /.])(0[1-9]|1[012])\2(0[1-9]|[12][0-9]|3[01])$`» will match „`1999-01-01`" but not "`1999/01-01`".

Again, how complex you want to make your regular expression depends on the data you are using it on, and how big a problem it is if an unwanted match slips through. If you are validating the user's input of a date in a script, it is probably easier to do certain checks outside of the regex. For example, excluding February 29th when the year is not a leap year is far easier to do in a scripting language. It is far easier to check if a year is divisible by 4 (and not divisible by 100 unless divisible by 400) using simple arithmetic than using regular expressions.

Here is how you could check a valid date in Perl. I also added round brackets to capture the year into a backreference.

```
sub isvaliddate {
  my $input = shift;
  if ($input =~ m!^((?:19|20)\d\d)[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])$!) {
    # At this point, $1 holds the year, $2 the month and $3 the day of the date entered
    if ($3 == 31 and ($2 == 4 or $2 == 6 or $2 == 9 or $2 == 11)) {
      return 0; # 31st of a month with 30 days
    } elsif ($3 >= 30 and $2 == 2) {
      return 0; # February 30th or 31st
    } elsif ($2 == 2 and $3 == 29 and not ($1 % 4 == 0 and ($1 % 100 != 0 or $1 % 400 == 0))) {
      return 0; # February 29th outside a leap year
    } else {
      return 1; # Valid date
    }
  } else {
    return 0; # Not a date
  }
}
```

To match a date in mm/dd/yyyy format, rearrange the regular expression to «`^(0[1-9]|1[012])[-/.](0[1-9]|[12][0-9]|3[01])[-/.](19|20)\d\d$`» . For dd-mm-yyyy format, use «`^(0[1-9]|[12][0-9]|3[01])[-/.](0[1-9]|1[012])[-/.](19|20)\d\d$`» . You can find additional variations of these regexes in RegexBuddy's library.

# 6. Finding or Verifying Credit Card Numbers

With a few simple regular expressions, you can easily verify whether your customer entered a valid credit card number on your order form. You can even determine the type of credit card being used. Each card issuer has its own range of card numbers, identified by the first 4 digits.

You can use a slightly different regular expression to find credit card numbers, or number sequences that might be credit card numbers, within larger documents. This can be very useful to prove in a security audit that you're not improperly exposing your clients' financial details.

We'll start with the order form.

## Stripping Spaces and Dashes

The first step is to remove all non-digits from the card number entered by the customer. Physical credit cards have spaces within the card number to group the digits, making it easier for humans to read or type in. So your order form should accept card numbers with spaces or dashes in them.

To remove all non-digits from the card number, simply use the "replace all" function in your scripting language to search for the regex «`[^0-9]+`» and replace it with nothing. If you only want to replace spaces and dashes, you could use «`[ -]+`». If this regex looks odd, remember that in a character class, the hyphen is a literal when it occurs right before the closing bracket (or right after the opening bracket or negating caret).

If you're wondering what the plus is for: that's for performance. If the input has consecutive non-digits, e.g. "`1===2`", then the regex will match the three equals signs at once, and delete them in one replacement. Without the plus, three replacements would be required. In this case, the savings are only a few microseconds. But it's a good habit to keep regex efficiency in the back of your mind. Though the savings are minimal here, so is the effort of typing the extra plus.

## Validating Credit Card Numbers on Your Order Form

Validating credit card numbers is the ideal job for regular expressions. They're just a sequence of 13 to 16 digits, with a few specific digits at the start that identify the card issuer. You can use the specific regular expressions below to alert customers when they try to use a kind of card you don't accept, or to route orders using different cards to different processors. All these regexes were taken from RegexBuddy's library.

- Visa: «`^4[0-9]{12}(?:[0-9]{3})?$`» All Visa card numbers start with a 4. New cards have 16 digits. Old cards have 13.
- MasterCard: «`^5[1-5][0-9]{14}$`» All MasterCard numbers start with the numbers 51 through 55. All have 16 digits.
- American Express: «`^3[47][0-9]{13}$`» American Express card numbers start with 34 or 37 and have 15 digits.
- Diners Club: «`^3(?:0[0-5]|[68][0-9])[0-9]{11}$`» Diners Club card numbers begin with 300 through 305, 36 or 38. All have 14 digits. There are Diners Club cards that begin with 5 and have 16 digits. These are a joint venture between Diners Club and MasterCard, and should be processed like a MasterCard.

- Discover: «^6(?:011|5[0-9]{2})[0-9]{12}$» Discover card numbers begin with 6011 or 65. All have 16 digits.
- JCB: «^(?:2131|1800|35\d{3})\d{11}$» JCB cards beginning with 2131 or 1800 have 15 digits. JCB cards beginning with 35 have 16 digits.

If you just want to check whether the card number looks valid, without determining the brand, you can combine the above six regexes into «^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})$». You'll see I've simply alternated all the regexes, and used a non-capturing group to put the anchors outside the alternation. You can easily delete the card types you don't accept from the list.

These regular expressions will easily catch numbers that are invalid because the customer entered too many or too few digits. They won't catch numbers with incorrect digits. For that, you need to follow the Luhn algorithm, which cannot be done with a regex. And of course, even if the number is mathematically valid, that doesn't mean a card with this number was issued or if there's money in the account. The benefit or the regular expression is that you can put it in a bit of JavaScript to instantly check for obvious errors, instead of making the customer wait 30 seconds for your credit card processor to fail the order. And if your card processor charges for failed transactions, you'll really want to implement both the regex and the Luhn validation.

## Finding Credit Card Numbers in Documents

With two simple modifications, you could use any of the above regexes to find card numbers in larger documents. Simply replace the caret and dollar with a word boundary, e.g.: «\b4[0-9]{12}(?:[0-9]{3})?\b».

If you're planning to search a large document server, a simpler regular expression will speed up the search. Unless your company uses 16-digit numbers for other purposes, you'll have few false positives. The regex «\b\d{13,16}\b» will find any sequence of 13 to 16 digits.

When searching a hard disk full of files, you can't strip out spaces and dashes first like you can when validating a single card number. To find card numbers with spaces or dashes in them, use «\b(?:\d[ -]*?){13,16}\b». This regex allows any amount of spaces and dashes anywhere in the number. This is really the only way. Visa and MasterCard put digits in sets of 4, while Amex and Discover use groups of 4, 5 and 6 digits. People typing in the numbers may have different ideas yet.

# 7. Matching Whole Lines of Text

Often, you want to match complete lines in a text file rather than just the part of the line that satisfies a certain requirement. This is useful if you want to delete entire lines in a search-and-replace in a text editor, or collect entire lines in an information retrieval tool.

To keep this example simple, let's say we want to match lines containing the word "John". The regex «`John`» makes it easy enough to locate those lines. But the software will only indicate „`John`" as the match, not the entire line containing the word.

The solution is fairly simple. To specify that we need an entire line, we will use the caret and dollar sign and turn on the option to make them match at embedded newlines. In software aimed at working with text files like EditPad Pro and PowerGREP, the anchors always match at embedded newlines. To match the parts of the line before and after the match of our original regular expression «`John`», we simply use the dot and the star. Be sure to turn *off* the option for the dot to match newlines.

The resulting regex is: «`^.*John.*$`». You can use the same method to expand the match of any regular expression to an entire line, or a block of complete lines. In some cases, such as when using alternation, you will need to group the original regex together using round brackets.

## Finding Lines Containing or Not Containing Certain Words

If a line can meet any out of series of requirements, simply use alternation in the regular expression. «`^.*\b(one|two|three)\b.*$`» matches a complete line of text that contains any of the words "one", "two" or "three". The first backreference will contain the word the line actually contains. If it contains more than one of the words, then the last (rightmost) word will be captured into the first backreference. This is because the star is greedy. If we make the first star lazy, like in «`^.*?\b(one|two|three)\b.*$`», then the backreference will contain the first (leftmost) word.

If a line must satisfy all of multiple requirements, we need to use lookahead. «`^(?=.*?\bone\b)(?=.*?\btwo\b)(?=.*?\bthree\b).*$`» matches a complete line of text that contains *all* of the words "one", "two" and "three". Again, the anchors must match at the start and end of a line and the dot must not match line breaks. Because of the caret, and the fact that lookahead is zero-width, all of the three lookaheads are attempted at the start of the each line. Each lookahead will match any piece of text on a single line («`.*?`») followed by one of the words. All three must match successfully for the entire regex to match. Note that instead of words like «`\bword\b`», you can put any regular expression, no matter how complex, inside the lookahead. Finally, «`.*$`» causes the regex to actually match the line, after the lookaheads have determined it meets the requirements.

If your condition is that a line should *not* contain something, use negative lookahead. «`^((?!regexp).)*$`» matches a complete line that does *not* match «`regexp`». Notice that unlike before, when using positive lookahead, I repeated both the negative lookahead and the dot together. For the positive lookahead, we only need to find one location where it can match. But the negative lookahead must be tested at each and every character position in the line. We must test that «`regexp`» fails everywhere, not just somewhere.

Finally, you can combine multiple positive and negative requirements as follows: «`^(?=.*?\bmust-have\b)(?=.*?\bmandatory\b)((?!avoid|illegal).)*$`» . When checking multiple positive requirements, the «`.*`» at the end of the regular expression full of zero-width assertions made sure that we

actually matched something. Since the negative requirement must match the entire line, it is easy to replace the «. *» with the negative test.

# 8. Deleting Duplicate Lines From a File

If you have a file in which all lines are sorted (alphabetically or otherwise), you can easily delete (consecutive) duplicate lines. Simply open the file in your favorite text editor, and do a search-and-replace searching for «`^(.*)(\r?\n\1)+$`» » matches a single-line string that does not allow the quote character to appear inside the string. Using the negated character class is more efficient than using a lazy dot. «`"[^"]*"`» allows the string to span across multiple lines.

«`"[^"\\\r\n]*(?:\\.[^"\\\r\n]*)*"`» matches a single-line string in which the quote character can appear if it is escaped by a backslash. Though this regular expression may seem more complicated than it needs to be, it is much faster than simpler solutions which can cause a whole lot of backtracking in case a double quote appears somewhere all by itself rather than part of a string. «`"[^"\\]*(?:\\.[^"\\]*)*"`» allows the string to span multiple lines.

You can adapt the above regexes to match any sequence delimited by two (possibly different) characters. If we use "`b`" for the starting character, "`e`" and the end, and "`x`" as the escape character, the version without escape becomes «`b[^e\r\n]*e`», and the version with escape becomes «`b[^ex\r\n]*(?:x.[^ex\r\n]*)*e`».

## Numbers

«`\b\d+\b`» matches a positive integer number. Do not forget the word boundaries! «`[-+]?\b\d+\b`» allows for a sign.

«`\b0[xX][0-9a-fA-F]+\b`» matches a C-style hexadecimal number.

«`((\b[0-9]+)?\.)?[0-9]+\b`» matches an integer number as well as a floating point number with optional integer part. «`(\b[0-9]+\.([0-9]+\b)?|\.[0-9]+\b)`» matches a floating point number with optional integer as well as optional fractional part, but does not match an integer number.

«`((\b[0-9]+)?\.)?\b[0-9]+([eE][-+]?[0-9]+)?\b`» matches a number in scientific notation. The mantissa can be an integer or floating point number with optional integer part. The exponent is optional.

«`\b[0-9]+(\.[0-9]+)?(e[+-]?[0-9]+)?\b`» also matches a number in scientific notation. The difference with the previous example is that if the mantissa is a floating point number, the integer part is mandatory.

If you read through the floating point number example, you will notice that the above regexes are different from what is used there. The above regexes are more stringent. They use word boundaries to exclude numbers that are part of other things like identifiers. You can prepend «`[-+]?`» to all of the above regexes to include an optional sign in the regex. I did not do so above because in programming languages, the + and - are usually considered operators rather than signs.

## Reserved Words or Keywords

Matching reserved words is easy. Simply use alternation to string them together: «`\b(first|second|third|etc)\b`» Again, do not forget the word boundaries.

# 10. Find Two Words Near Each Other

Some search tools that use boolean operators also have a special operator called "near". Searching for “term1 near term2” finds all occurrences of term1 and term2 that occur within a certain "distance" from each other. The distance is a number of words. The actual number depends on the search tool, and is often configurable.

You can easily perform the same task with the proper regular expression.

## Emulating “near” with a Regular Expression

With regular expressions you can describe almost any text pattern, including a pattern that matches two words near each other. This pattern is relatively simple, consisting of three parts: the first word, a certain number of unspecified words, and the second word. An unspecified word can be matched with the shorthand character class «`\w+`». The spaces and other characters between the words can be matched with «`\W+`» (uppercase W this time).

The complete regular expression becomes «`\bword1\W+(?:\w+\W+){1,6}?word2\b`» . The quantifier «`{1,6}?`» makes the regex require at least one word between “word1” and “word2”, and allow at most six words.

If the words may also occur in reverse order, we need to specify the opposite pattern as well: «`\b(?:word1\W+(?:\w+\W+){1,6}?word2|word2\W+(?:\w+\W+){1,6}?word1)\b`»

If you want to find any pair of two words out of a list of words, you can use: «`\b(word1|word2|word3)(?:\W+\w+){1,6}?\W+(word1|word2|word3)\b`». This regex will also find a word near itself, e.g. it will match „`word2 near word2`”.

# 11. Runaway Regular Expressions: Catastrophic Backtracking

Consider the regular expression «(x+x+)+y». Before you scream in horror and say this contrived example should be written as «xx+y» to match exactly the same without those terribly nested quantifiers: just assume that each "x" represents something more complex, with certain strings being matched by both "x". See the section on HTML files below for a real example.

Let's see what happens when you apply this regex to "xxxxxxxxxxy". The first «x+» will match all 10 „x" characters. The second «x+» fails. The first «x+» then backtracks to 9 matches, and the second one picks up the remaining „x". The group has now matched once. The group repeats, but fails at the first «x+». Since one repetition was sufficient, the group matches. «y» matches „y" and an overall match is found. The regex is declared functional, the code is shipped to the customer, and his computer explodes. Almost.

The above regex turns ugly when the "y" is missing from the subject string. When «y» fails, the regex engine backtracks. The group has one iteration it can backtrack into. The second «x+» matched only one „x", so it can't backtrack. But the first «x+» can give up one "x". The second «x+» promptly matches „xx". The group again has one iteration, fails the next one, and the «y» fails. Backtracking again, the second «x+» now has one backtracking position, reducing itself to match „x". The group tries a second iteration. The first «x+» matches but the second is stuck at the end of the string. Backtracking again, the first «x+» in the group's first iteration reduces itself to 7 characters. The second «x+» matches „xxx". Failing «y», the second «x+» is reduced to „xx" and then „x". Now, the group can match a second iteration, with one „x" for each «x+». But this (7,1),(1,1) combination fails too. So it goes to (6,4) and then (6,2)(1,1) and then (6,1),(2,1) and then (6,1),(1,2) and then I think you start to get the drift.

If you try this regex on a 10x string in RegexBuddy's debugger, it'll take 2558 steps to figure out the final «y» is missing. For an 11x string, it needs 5118 steps. For 12, it takes 10238 steps. Clearly we have an exponential complexity of O(2^n) here. At 21x the debugger bows out at 2.8 million steps, diagnosing a bad case of catastrophic backtracking.

RegexBuddy is forgiving in that it detects it's going in circles, and aborts the match attempt. Other regex engines (like .NET) will keep going forever, while others will crash with a stack overflow (like Perl, before version 5.10). Stack overflows are particularly nasty on Windows, since they tend to make your application vanish without a trace or explanation. Be very careful if you run a web service that allows users to supply their own regular expressions. People with little regex experience have surprising skill at coming up with exponentially complex regular expressions.

## Possessive Quantifiers and Atomic Grouping to The Rescue

In the above example, the sane thing to do is obviously to rewrite it as «xx+y» which eliminates the nested quantifiers entirely. Nested quantifiers are repeated or alternated tokens inside a group that is itself repeated or alternated. These almost always lead to catastrophic backtracking. About the only situation where they don't is when the start of each alternative inside the group is not optional, and mutually exclusive with the start of all the other alternatives, and mutually exclusive with the token that follows it (inside its alternative inside the group). E.g. «(a+b+|c+d+)+y» is safe. If anything fails, the regex engine will backtrack through the whole regex, but it will do so linearly. The reason is that all the tokens are mutually exclusive. None of them can match any characters matched by any of the others. So the match attempt at each backtracking

position will fail, causing the regex engine to backtrack linearly. If you test this on "aaaabbbbccccdddd", RegexBuddy needs only 13 steps rather than millions of steps to figure it out.

However, it's not always possible or easy to rewrite your regex to make everything mutually exclusive. So we need a way to tell the regex engine not to backtrack. When we've grabbed all the x's, there's no need to backtrack. There couldn't possibly be a "y" in anything matched by either «x+». Using a possessive quantifier, our regex becomes «(x+x+)++y». This fails the 21x string in merely 7 steps. That's 6 steps to match all the x's, and 1 step to figure out that «y» fails. Almost no backtracking is done. Using an atomic group, the regex becomes «(?>(x+x+)+)y» with the exact same results.

## A Real Example: Matching CSV Records

Here's a real example from a technical support case I once handled. The customer was trying to find lines in a comma-delimited text file where the 12th item on a line started with a "P". He was using the innocently-looking regexp «^(.*?,){11}P» .

At first sight, this regex looks like it should do the job just fine. The lazy dot and comma match a single comma-delimited field, and the {11} skips the first 11 fields. Finally, the P checks if the 12th field indeed starts with P. In fact, this is exactly what will happen when the 12th field indeed starts with a P.

The problem rears its ugly head when the 12th field does not start with a P. Let's say the string is "1,2,3,4,5,6,7,8,9,10,11,12,13". At that point, the regex engine will backtrack. It will backtrack to the point where «^(.*?,){11}» had consumed „1,2,3,4,5,6,7,8,9,10,11", giving up the last match of the comma. The next token is again the dot. The dot matches a comma. *The dot matches the comma!* However, the comma does not match the "1" in the 12th field, so the dot continues until the 11th iteration of «.*?,» has consumed „11,12,". You can already see the root of the problem: the part of the regex (the dot) matching the contents of the field also matches the delimiter (the comma). Because of the double repetition (star inside {11}), this leads to a catastrophic amount of backtracking.

The regex engine now checks whether the 13th field starts with a P. It does not. Since there is no comma after the 13th field, the regex engine can no longer match the 11th iteration of «.*?,». But it does not give up there. It backtracks to the 10th iteration, expanding the match of the 10th iteration to „10,11,". Since there is still no P, the 10th iteration is expanded to „10,11,12,". Reaching the end of the string again, the same story starts with the 9th iteration, subsequently expanding it to „9,10,", „9,10,11,", „9,10,11,12,". But between each expansion, there are more possibilities to be tried. When the 9th iteration consumes „9,10,", the 10th could match just „11," as well as „11,12,". Continuously failing, the engine backtracks to the 8th iteration, again trying all possible combinations for the 9th, 10th, and 11th iterations.

You get the idea: the possible number of combinations that the regex engine will try for each line where the 12th field does not start with a P is huge. All this would take a long time if you ran this regex on a large CSV file where most rows don't have a P at the start of the 12th field.

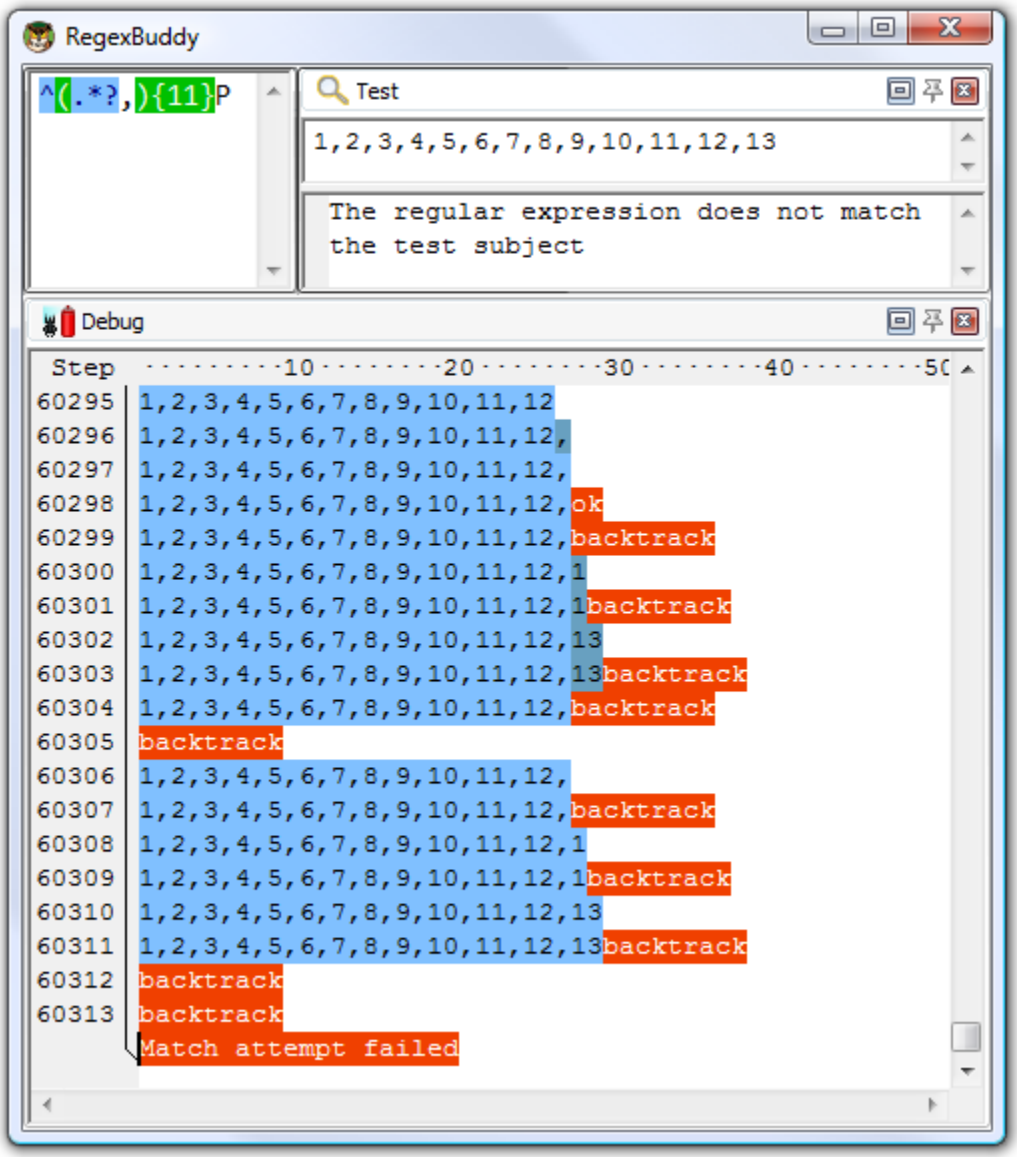
## Preventing Catastrophic Backtracking

The solution is simple. When nesting repetition operators, make absolutely sure that there is only one way to match the same match. If repeating the inner loop 4 times and the outer loop 7 times results in the same overall match as repeating the inner loop 6 times and the outer loop 2 times, you can be sure that the regex engine will try all those combinations.

In our example, the solution is to be more exact about what we want to match. We want to match 11 comma-delimited fields. The fields must not contain comma's. So the regex becomes: «`^([^,\r\n]*,){11}P`» . If the P cannot be found, the engine will still backtrack. But it will backtrack only 11 times, and each time the «`[^,\r\n]`» is not able to expand beyond the comma, forcing the regex engine to the previous one of the 11 iterations immediately, without trying further options.

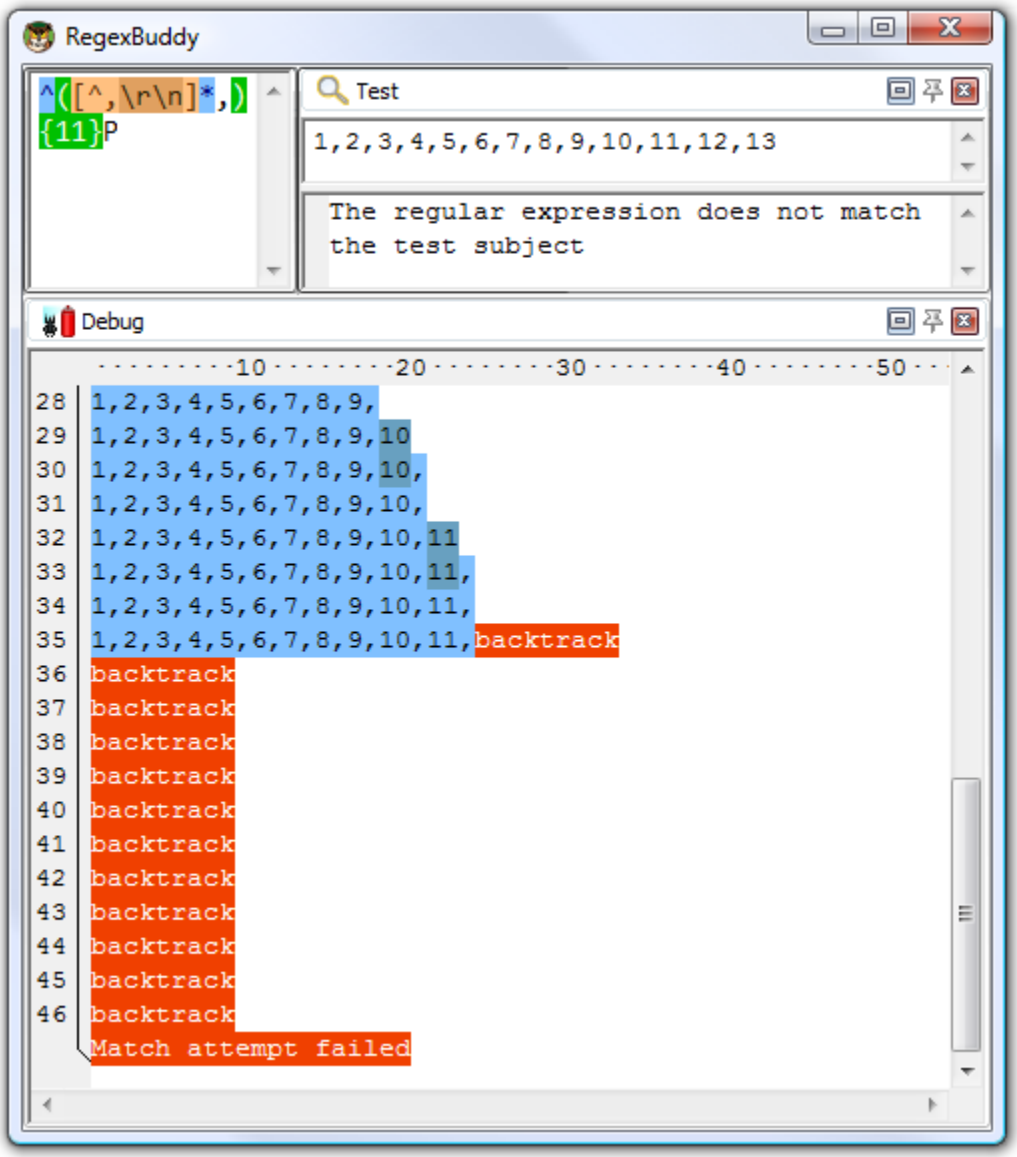
## See the Difference with RegexBuddy

If you try this example with RegexBuddy's debugger, you will see that the original regex «`^(.*?,){11}P`» needs 29,685 steps to conclude there regex cannot match "`1,2,3,4,5,6,7,8,9,10,11,12`". If the string is "`1,2,3,4,5,6,7,8,9,10,11,12,13`", just 3 characters more, the number of steps doubles to 60,313. It's not too hard to imagine that at this kind of exponential rate, attempting this regex on a large file with long lines could easily take forever.



Our improved regex «^([^,\r\n]*,){11}P», however, needs just forty-eight steps to fail, whether the subject string has 12 numbers, 13 numbers, 16 numbers or a billion. While the complexity of the original regex was exponential, the complexity of the improved regex is constant with respect to whatever follows the 12th field. The reason is the regex fails immediately when it discovers the 12th field doesn't start with a P. It simply backtracks 12 times without expanding again, and that's it.

The complexity of the improved regex is linear to the length of the first 11 fields. 36 steps are needed in our example. That's the best we can do, since the engine does have to scan through all the characters of the first 11 fields to find out where the 12th one begins. Our improved regex is a perfect solution.



## Alternative Solution Using Atomic Grouping

In the above example, we could easily reduce the amount of backtracking to a very low level by better specifying what we wanted. But that is not always possible in such a straightforward manner. In that case, you should use atomic grouping to prevent the regex engine from backtracking.

Using atomic grouping, the above regex becomes «`^(?>(.*?,){11})P`». Everything between `(?>)` is treated as one single token by the regex engine, once the regex engine leaves the group. Because the entire group is one token, no backtracking can take place once the regex engine has found a match for the group. If backtracking is required, the engine has to backtrack to the regex token before the group (the caret in our example). If there is no token before the group, the regex must retry the entire regex at the next position in the string.

Let's see how «`^(?>(.*?,){11})P`» is applied to "`1,2,3,4,5,6,7,8,9,10,11,12,13`". The caret matches at the start of the string and the engine enters the atomic group. The star is lazy, so the dot is initially skipped. But the comma does not match "`1`", so the engine backtracks to the dot. That's right: backtracking is allowed here. The star is not possessive, and is not immediately enclosed by an atomic group. That is, the regex engine did not cross the closing round bracket of the atomic group. The dot matches „`1`", and the comma matches too. «`{11}`» causes further repetition until the atomic group has matched „`1,2,3,4,5,6,7,8,9,10,11,`".

Now, the engine leaves the atomic group. Because the group is atomic, all backtracking information is discarded and the group is now considered a single token. The engine now tries to match «`P`» to the "`1`" in the 12th field. This fails.

So far, everything happened just like in the original, troublesome regular expression. Now comes the difference. «`P`» failed to match, so the engine backtracks. The previous token is an atomic group, so the group's entire match is discarded and the engine backtracks further to the caret. The engine now tries to match the caret at the next position in the string, which fails. The engine walks through the string until the end, and declares failure. Failure is declared after 30 attempts to match the caret, and just one attempt to match the atomic group, rather than after 30 attempts to match the caret and a huge number of attempts to try all combinations of both quantifiers in the regex.

That is what atomic grouping and possessive quantifiers are for: efficiency by disallowing backtracking. The most efficient regex for our problem at hand would be «`^(?>((?>[^,\r\n]*),){11})P`» , since possessive, greedy repetition of the star is faster than a backtracking lazy dot. If possessive quantifiers are available, you can reduce clutter by writing «`^(?>([^,\r\n]*+,){11})P`» .

## Quickly Matching a Complete HTML File

Another common situation where catastrophic backtracking occurs is when trying to match "something" followed by "anything" followed by "another something" followed by "anything", where the lazy dot «`.*?`» is used. The more "anything", the more backtracking. Sometimes, the lazy dot is simply a symptom of a lazy programmer. «`".*?"`» is not appropriate to match a double-quoted string, since you don't really want to allow anything between the quotes. A string can't have (unescaped) embedded quotes, so «`"[^"\r\n]*"`» is more appropriate, and won't lead to catastrophic backtracking when combined in a larger regular expression. However, sometimes "anything" really is just that. The problem is that "another something" also qualifies as "anything", giving us a genuine «`x+x+`» situation.

Suppose you want to use a regular expression to match a complete HTML file, and extract the basic parts from the file. If you know the structure of HTML files, writing the regex «<html>.*?<head>.*?<title>.*?</title>.*?</head>.*?<body[^>]*>.*?</body>.*?</html>» is very straight-forward. With the "dot matches newlines" or "single line" matching mode turned on, it will work just fine on valid HTML files.

Unfortunately, this regular expression won't work nearly as well on an HTML file that misses some of the tags. The worst case is a missing </html> tag at the end of the file. When «</html>» fails to match, the regex engine backtracks, giving up the match for «</body>.*?». It will then further expand the lazy dot before «</body>», looking for a second closing "</body>" tag in the HTML file. When that fails, the engine gives up «<body[^>]*>.*?», and starts looking for a second opening "<body[^>]*>" tag all the way to the end of the file. Since that also fails, the engine proceeds looking all the way to the end of the file for a second closing head tag, a second closing title tag, etc.

If you run this regex in RegexBuddy's debugger, the output will look like a sawtooth. The regex matches the whole file, backs up a little, matches the whole file again, backs up some more, backs up yet some more, matches everything again, etc. until each of the 7 «.*?» tokens has reached the end of the file. The result is that this regular has a worst case complexity of N^7. If you double the length of the HTML file with the missing <html> tag by appending text at the end, the regular expression will take 128 times (2^7) as long to figure out the HTML file isn't valid. This isn't quite as disastrous as the 2^N complexity of our first example, but will lead to very unacceptable performance on larger invalid files.

In this situation, we know that each of the literal text blocks in our regular expression (the HTML tags, which function as delimiters) will occur only once in a valid HTML file. That makes it very easy to package each of the lazy dots (the delimited content) in an atomic group.

«<html>(?>.*?<head>)(?>.*?<title>)(?>.*?</title>)(?>.*?</head>)(?>.*?<body[^>]*>)(?>.*?</body>).*?</html>» will match a valid HTML file in the same number of steps as the original regex. The gain is that it will fail on an invalid HTML file almost as fast as it matches a valid one. When «</html>» fails to match, the regex engine backtracks, giving up the match for the last lazy dot. But then, there's nothing further to backtrack to. Since all of the lazy dots are in an atomic group, the regex engines has discarded their backtracking positions. The groups function as a "do not expand further" roadblock. The regex engine is forced to announce failure immediately.

I'm sure you've noticed that each atomic group also contains an HTML tag after the lazy dot. This is critical. We do allow the lazy dot to backtrack until its matching HTML tag was found. E.g. when «.*?</body>» is processing "Last paragraph</p></body>", the «</» regex tokens will match „</" in "</p>". However, «b» will fail "p". At that point, the regex engine will backtrack and expand the lazy dot to include „</p>". Since the regex engine hasn't left the atomic group yet, it is free to backtrack inside the group. Once «</body>» has matched, and the regex engine leaves the atomic group, it discards the lazy dot's backtracking positions. Then it can no longer be expanded.

Essentially, what we've done is to bind a repeated regex token (the lazy dot to match HTML content) to the non-repeated regex token that follows it (the literal HTML tag). Since anything, including HTML tags, can appear between the HTML tags in our regular expression, we cannot use a negated character class instead of the lazy dot to prevent the delimiting HTML tags from being matched as HTML content. But we can and did achieve the same result by combining each lazy dot and the HTML tag following it into an atomic group. As soon as the HTML tag is matched, the lazy dot's match is locked down. This ensures that the lazy dot will never match the HTML tag that should be matched by the literal HTML tag in the regular expression.

# 12. Repeating a Capturing Group vs. Capturing a Repeated Group

When creating a regular expression that needs a capturing group to grab part of the text matched, a common mistake is to repeat the capturing group instead of capturing a repeated group. The difference is that the repeated capturing group will capture only the last iteration, while a group capturing another group that's repeated will capture all iterations. An example will make this clear.

Let's say you want to match a tag like „`!abc!`" or „`!123!`". Only these two are possible, and you want to capture the „`abc`" or „`123`" to figure out which tag you got. That's easy enough: «`!(abc|123)!`» will do the trick.

Now let's say that the tag can contain multiple sequences of "`abc`" and "`123`", like „`!abc123!`" or „`!123abcabc!`". The quick and easy solution is «`!(abc|123)+!`». This regular expression will indeed match these tags. However, it no longer meets our requirement to capture the tag's label into the capturing group. When this regex matches „`!abc123!`", the capturing group stores only „`123`". When it matches „`!123abcabc!`", it only stores „`abc`".

This is easy to understand if we look at how the regex engine applies «`!(abc|123)+!`» to "`!abc123!`". First, «`!`» matches „`!`". The engine then enters the capturing group. It makes note that capturing group #1 was entered when the engine reached the position between the first and second character in the subject string. The first token in the group is «`abc`», which matches „`abc`". A match is found, so the second alternative isn't tried. (The engine does store a backtracking position, but this won't be used in this example.) The engine now leaves the capturing group. It makes note that capturing group #1 was exited when the engine reached the position between the 4th and 5th characters in the string.

After having exited from the group, the engine notices the plus. The plus is greedy, so the group is tried again. The engine enters the group again, and takes note that capturing group #1 was entered between the 4th and 5th characters in the string. It also makes note that since the plus is not possessive, it may be backtracked. That is, if the group cannot be matched a second time, that's fine. In this backtracking note, the regex engine also saves the entrance and exit positions of the group during the previous iteration of the group.

«`abc`» fails to match "`123`", but «`123`» succeeds. The group is exited again. The exit position between characters 7 and 8 is stored.

The plus allows for another iteration, so the engine tries again. Backtracking info is stored, and the new entrance position for the group is saved. But now, both «`abc`» and «`123`» fail to match "`!`". The group fails, and the engine backtracks. While backtracking, the engine restores the capturing positions for the group. Namely, the group was entered between characters 4 and 5, and existed between characters 7 and 8.

The engine proceeds with «`!`», which matches „`!`". An overall match is found. The overall match spans the whole subject string. The capturing group spaces characters 5, 6 and 7, or „`123`". Backtracking information is discarded when a match is found, so there's no way to tell after the fact that the group had a previous iteration that matched „`abc`". (The only exception to this is the .NET regex engine, which does preserve backtracking information for capturing groups after the match attempt.)

The solution to capturing „`abc123`" in this example should be obvious now: the regex engine should enter and leave the group only once. This means that the plus should be inside the capturing group rather than outside. Since we do need to group the two alternatives, we'll need to place a second capturing group around

the repeated group: «`!((abc|123)+)!`». When this regex matches „`!abc123!`”, capturing group #1 will store „`abc123`”, and group #2 will store „`123`”. Since we’re not interested in the inner group’s match, we can optimize this regular expression by making the inner group non-capturing: «`!((?:abc|123)+)!`».

# 13. Mixing Unicode and 8-bit Character Codes

Internally, computers deal with numbers, not with characters. When you save a text file, each character is mapped to a number, and the numbers are stored on disk. When you open a text file, the numbers are read and mapped back to characters. When processing text with a regular expression, the regular expression needs to use the same mapping as you used to create the file or string you want the regex to process.

When you simply type in all the characters in your regular expression, you normally don't have anything to worry about. The application or programming library that provides the regular expression functionality will know what text encodings your subject string uses, and process it accordingly. So if you want to search for the euro currency symbol, and you have a European keyboard, just press AltGr+E. Your regex «€» will find all euro symbols just fine.

But you can't press AltGr+E on a US keyboard. Or perhaps you like your source code to be 7-bit clean (i.e. plain ASCII). In those cases, you'll need to use a character escape in your regular expression.

If your regular expression engine supports Unicode, simply use the Unicode escape «`\u20AC`» (most Unicode flavors) or «`\x{20AC}`» (Perl and PCRE). U+20AC is the Unicode code point for the euro symbol. It will always match the euro symbol, whether your subject string is encoded in UTF-8, UTF-16, UCS-2 or whatever. Even when your subject string is encoded with a legacy 8-bit code page, there's no confusion. You may need to tell the application or regex engine what encoding your file uses. But «`\u20AC`» is always the euro symbol.

Most Unicode regex engines also support the 8-bit character escape «`\xFF`». However, its use is not recommended. For characters «`\x00`» through «`\x7F`», there's usually no trouble. The first 128 Unicode code points are identical to the ASCII table that most 8-bit code pages are based on.

But the interpretation of «`\x80`» and above may vary. A pure Unicode engine will treat this identical to «`\u0080`», which represents a Latin-1 control code. But what most people expect is that «`\x80`» matches the euro symbol, as that occupies position 80h in all Windows code pages. And it will when using an 8-bit regex engine if your text file is encoded using a Windows code page.

Since most people expect «`\x80`» to be treated as an 8-bit character rather than the Unicode code point «`\u0080`», some Unicode regex engines do exactly that. Some are hard-wired to use a particular code page, say Windows 1252 or your computer's default code page, to interpret 8-bit character codes.

Other engines will let it depend on the input string. E.g. the JGsoft engine will treat «`\x80`» as «`\u0080`» when searching through a Unicode text file, but as «`\u20AC`» when searching through a Windows 1252 text file. There's no magic here. It matches the character with index 80h in the text file, regardless of the text file's encoding. Unicode code point U+0080 is a Latin-1 control code, while Windows 1252 character index 80h is the euro symbol. In reverse, if you type in the euro symbol in a text editor, saving it as UTF-16 will save two bytes `AC 20`, while saving as Windows 1252 will give you one byte `80`.

If you find the above confusing, simply don't use «`\x80`» through «`\xFF`» with a regex engine that supports Unicode.

## 8-bit Regex Engines

When working with a legacy (obsolete?) regular expression engine that works on 8-bit data only, you can't use Unicode escapes like «`\u20AC`». «`\x80`» is all you have. Note that even modern engines have legacy modes. E.g. the popular regex library PCRE runs as an 8-bit engine by default. You need to explicitly enable UTF-8 support if you want to use Unicode features. When you do, PCRE also expects you to convert your subject strings to UTF-8.

When crafting a regular expression for an 8-bit engine, you'll have to take into account which character set or code page you'll be working with. 8-bit regex engines just don't care. If you type «`\x80`» into your regex, it will match any byte 80h, regardless of what that byte represents. That'll be the euro symbol in a Windows 1252 text file, a control code in a Latin-1 file, and the digit zero in an EBCDIC file.

Even for literal characters in your regex, you'll have to match up the encoding you're using in the regular expression with the subject encoding. If your application is using the Latin-1 code page, and you use the regex «À», it'll match «Ŕ» when you search through a Latin-2 text file. The application would duly display this as „À" on the screen, because it's using the wrong code page. This problem is not really specific to regular expressions. You'll encounter it any time you're working with files and applications that use different 8-bit encodings.

So when working with 8-bit data, open the actual data you're working with in a hex editor. See the bytes being used, and specify those in your regular expression.

Where it gets really hairy is if you're processing Unicode files with an 8-bit engine. Let's go back to our text file with just a euro symbol. When saved as little endian UTF-16 (called "Unicode" on Windows), an 8-bit regex engine will see two bytes `AC 20` (remember that little endian reverses the bytes). When saved as UTF-8 (which has no endianness), our 8-bit engine will see three bytes `E2 82 AC`. You'd need «`\xE2\x82\xAC`» to match the euro symbol in an UTF-8 file with an 8-bit regex engine.

# Part 4

# Regular Expression Reference

# 1. Basic Syntax Reference

## Characters

Character: Any character except `[\^$.|?*+()`
Description: All characters except the listed special characters match a single instance of themselves. `{` and `}` are literal characters, unless they're part of a valid regular expression token (e.g. the `{n}` quantifier).
Example: «`a`» matches „a"

Character: `\` (backslash) followed by any of `[\^$.|?*+(){}`
Description: A backslash escapes special characters to suppress their special meaning.
Example: «`\+`» matches „+"

Character: `\Q...\E`
Description: Matches the characters between `\Q` and `\E` literally, suppressing the meaning of special characters.
Example: «`\Q+-*/\E`» matches „+-*/"

Character: `\xFF` where FF are 2 hexadecimal digits
Description: Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.
Example: «`\xA9`» matches „©" when using the Latin-1 code page.

Character: `\n`, `\r` and `\t`
Description: Match an LF character, CR character and a tab character respectively. Can be used in character classes.
Example: «`\r\n`» matches a DOS/Windows CRLF line break.

Character: `\a`, `\e`, `\f` and `\v`
Description: Match a bell character (`\x07`), escape character (`\x1B`), form feed (`\x0C`) and vertical tab (`\x0B`) respectively. Can be used in character classes.
Character: `\cA` through `\cZ`
Description: Match an ASCII character Control+A through Control+Z, equivalent to «`\x01`» through «`\x1A`». Can be used in character classes.
Example: «`\cM\cJ`» matches a DOS/Windows CRLF line break.

## Character Classes or Character Sets [abc]

Character: `[` (opening square bracket)
Description: Starts a character class. A character class matches a single character out of all the possibilities offered by the character class. Inside a character class, different rules apply. The rules in this section are only valid inside character classes. The rules outside this section are not valid in character classes, except for a few character escapes that are indicated with "can be used inside character classes".
Character: Any character except `^-]\` add that character to the possible matches for the character class.
Description: All characters except the listed special characters.
Example: «`[abc]`» matches „`a`", „`b`" or „`c`"

Character: `\` (backslash) followed by any of `^-]\`
Description: A backslash escapes special characters to suppress their special meaning.
Example: «`[\^\]]`» matches „`^`" or „`]`"

Character: `-` (hyphen) except immediately after the opening `[`
Description: Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening `[`)
Example: «`[a-zA-Z0-9]`» matches any letter or digit

Character: `^` (caret) immediately after the opening `[`
Description: Negates the character class, causing it to match a single character *not* listed in the character class. (Specifies a caret if placed anywhere except after the opening `[`)
Example: «`[^a-d]`» matches „`x`" (any character except a, b, c or d)

Character: `\d`, `\w` and `\s`
Description: Shorthand character classes matching digits, word characters (letters, digits, and underscores), and whitespace (spaces, tabs, and line breaks). Can be used inside and outside character classes.
Example: «`[\d\s]`» matches a character that is a digit or whitespace

Character: `\D`, `\W` and `\S`
Description: Negated versions of the above. Should be used only outside character classes. (Can be used inside, but that is confusing.)
Example: «`\D`» matches a character that is not a digit

Character: `[\b]`
Description: Inside a character class, `\b` is a backspace character.
Example: «`[\b\t]`» matches a backspace or tab character

## Dot

Character: `.` (dot)
Description: Matches any single character except line break characters \r and \n. Most regex flavors have an option to make the dot match line break characters too.
Example: «`.`» matches „`x`" or (almost) any other character

## Anchors

Character: ^ (caret)
Description: Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the caret match after line breaks (i.e. at the start of a line in a file) as well.
Example: «^.» matches „a" in "abc\ndef". Also matches „d" in "multi-line" mode.

Character: $ (dollar)
Description: Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the dollar match before line breaks (i.e. at the end of a line in a file) as well. Also matches before the very last line break if the string ends with a line break.
Example: «.$» matches „f" in "abc\ndef". Also matches „c" in "multi-line" mode.

Character: \A
Description: Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Never matches after line breaks.
Example: «\A.» matches „a" in "abc"

Character: \Z
Description: Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks, except for the very last line break if the string ends with a line break.
Example: «.\Z» matches „f" in "abc\ndef"

Character: \z
Description: Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks.
Example: «.\z» matches „f" in "abc\ndef"

## Word Boundaries

Character: \b
Description: Matches at the position between a word character (anything matched by «\w») and a non-word character (anything matched by «[^\w]» or «\W») as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.
Example: «.\b» matches „c" in "abc"

Character: \B
Description: Matches at the position between two word characters (i.e the position between «\w\w») as well as at the position between two non-word characters (i.e. «\W\W»).
Example: «\B.\B» matches „b" in "abc"

## Alternation

Character: | (pipe)
Description: Causes the regex engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options.
Example: «`abc|def|xyz`» matches „`abc`", „`def`" or „`xyz`"

Character: | (pipe)
Description: The pipe has the lowest precedence of all operators. Use grouping to alternate only part of the regular expression.
Example: «`abc(def|xyz)`» matches „`abcdef`" or „`abcxyz`"

## Quantifiers

Character: ? (question mark)
Description: Makes the preceding item optional. Greedy, so the optional item is included in the match if possible.
Example: «`abc?`» matches „`ab`" or „`abc`"

Character: ??
Description: Makes the preceding item optional. Lazy, so the optional item is excluded in the match if possible. This construct is often excluded from documentation because of its limited use.
Example: «`abc??`» matches „`ab`" or „`abc`"

Character: * (star)
Description: Repeats the previous item zero or more times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.
Example: «`".*"`» matches „`"def" "ghi"`" in "`abc "def" "ghi" jkl`"

Character: *? (lazy star)
Description: Repeats the previous item zero or more times. Lazy, so the engine first attempts to skip the previous item, before trying permutations with ever increasing matches of the preceding item.
Example: «`".*?"`» matches „`"def"`" in "`abc "def" "ghi" jkl`"

Character: + (plus)
Description: Repeats the previous item once or more. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.
Example: «`".+"`» matches „`"def" "ghi"`" in "`abc "def" "ghi" jkl`"

Character: +? (lazy plus)
Description: Repeats the previous item once or more. Lazy, so the engine first matches the previous item only once, before trying permutations with ever increasing matches of the preceding item.
Example: «`".+?"`» matches „`"def"`" in "`abc "def" "ghi" jkl`"

Character: `{n}` where n is an integer >= 1
Description: Repeats the previous item exactly n times.
Example: «`a{3}`» matches „`aaa`"

Character: `{n,m}` where n >= 0 and m >= n
Description: Repeats the previous item between n and m times. Greedy, so repeating m times is tried before reducing the repetition to n times.
Example: «`a{2,4}`» matches „`aaaa`", „`aaa`" or „`aa`"

Character: `{n,m}?` where n >= 0 and m >= n
Description: Repeats the previous item between n and m times. Lazy, so repeating n times is tried before increasing the repetition to m times.
Example: «`a{2,4}?`» matches „`aa`", „`aaa`" or „`aaaa`"

Character: `{n,}` where n >= 0
Description: Repeats the previous item at least n times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only n times.
Example: «`a{2,}`» matches „`aaaaa`" in "`aaaaa`"

Character: `{n,}?` where n >= 0
Description: Repeats the previous item n or more times. Lazy, so the engine first matches the previous item n times, before trying permutations with ever increasing matches of the preceding item.
Example: «`a{2,}?`» matches „`aa`" in "`aaaaa`"

# 2. Advanced Syntax Reference

## Grouping and Backreferences

Character: `(regex)`
Description: Round brackets group the regex between them. They capture the text matched by the regex inside them that can be reused in a backreference, and they allow you to apply regex operators to the entire grouped regex.
Example: «`(abc){3}`» matches „`abcabcabc`". First group matches „`abc`".

Character: `(?:regex)`
Description: Non-capturing parentheses group the regex so you can apply regex operators, but do not capture anything and do not create backreferences.
Example: «`(?:abc){3}`» matches „`abcabcabc`". No groups.

Character: `\1` through `\9`
Description: Substituted with the text matched between the 1st through 9th pair of capturing parentheses. Some regex flavors allow more than 9 backreferences.
Example: «`(abc|def)=\1`» matches „`abc=abc`" or „`def=def`", but not "`abc=def`" or "`def=abc`".

## Modifiers

Character: `(?i)`
Description: Turn on case insensitivity for the remainder of the regular expression. (Older regex flavors may turn it on for the entire regex.)
Example: «`te(?i)st`» matches „`teST`" but not "`TEST`".

Character: `(?-i)`
Description: Turn off case insensitivity for the remainder of the regular expression.
Example: «`(?i)te(?-i)st`» matches „`TEst`" but not "`TEST`".

Character: `(?s)`
Description: Turn on "dot matches newline" for the remainder of the regular expression. (Older regex flavors may turn it on for the entire regex.)
Character: `(?-s)`
Description: Turn off "dot matches newline" for the remainder of the regular expression.
Character: `(?m)`
Description: Caret and dollar match after and before newlines for the remainder of the regular expression. (Older regex flavors may apply this to the entire regex.)
Character: `(?-m)`
Description: Caret and dollar only match at the start and end of the string for the remainder of the regular expression.
Character: `(?x)`
Description: Turn on free-spacing mode to ignore whitespace between regex tokens, and allow # comments.
Character: `(?-x)`
Description: Turn off free-spacing mode.
Character: `(?i-sm)`
Description: Turns on the option "i" and turns off "s" and "m" for the remainder of the regular expression. (Older regex flavors may apply this to the entire regex.)
Character: `(?i-sm:regex)`
Description: Matches the regex inside the span with the option "i" turned on and "m" and "s" turned off.
Example: «`(?i:te)st`» matches „`TEst`" but not "`TEST`".

## Atomic Grouping and Possessive Quantifiers

Character: `(?>regex)`
Description: Atomic groups prevent the regex engine from backtracking back into the group (forcing the group to discard part of its match) after a match has been found for the group. Backtracking can occur inside the group before it has matched completely, and the engine can backtrack past the entire group, discarding its match entirely. Eliminating needless backtracking provides a speed increase. Atomic grouping is often indispensable when nesting quantifiers to prevent a catastrophic amount of backtracking as the engine needlessly tries pointless permutations of the nested quantifiers.
Example: «`x(?>\w+)x`» is more efficient than «`x\w+x`» if the second x cannot be matched.

Character: `?+`, `*+`, `++` and `{m,n}+`
Description: Possessive quantifiers are a limited yet syntactically cleaner alternative to atomic grouping. Only available in a few regex flavors. They behave as normal greedy quantifiers, except that they will not give up part of their match for backtracking.
Example: «`x++`» is identical to «`(?>x+)`»

## Lookaround

Character: `(?=regex)`
Description: Zero-width positive lookahead. Matches at a position where the pattern inside the lookahead can be matched. Matches only the position. It does not consume any characters or expand the match. In a pattern like «`one(?=two)three`», both «`two`» and «`three`» have to match at the position where the match of «`one`» ends.
Example: «`t(?=s)`» matches the second „`t`" in „`streets`".

Character: `(?!regex)`
Description: Zero-width negative lookahead. Identical to positive lookahead, except that the overall match will only succeed if the regex inside the lookahead fails to match.
Example: «`t(?!s)`» matches the first „`t`" in „`streets`".

Character: `(?<=regex)`
Description: Zero-width positive lookbehind. Matches at a position if the pattern inside the lookahead can be matched ending at that position (i.e. to the left of that position). Depending on the regex flavor you're using, you may not be able to use quantifiers and/or alternation inside lookbehind.
Example: «`(?<=s)t`» matches the first „`t`" in „`streets`".

Character: `(?<!regex)`
Description: Zero-width negative lookbehind. Matches at a position if the pattern inside the lookahead cannot be matched ending at that position.
Example: «`(?<!s)t`» matches the second „`t`" in „`streets`".

## Continuing from The Previous Match

Character: `\G`
Description: Matches at the position where the previous match ended, or the position where the current match attempt started (depending on the tool or regex flavor). Matches at the start of the string during the first match attempt.
Example: «`\G[a-z]`» first matches „`a`", then matches „`b`" and then fails to match in "`ab_cd`".

## Conditionals

Character: `(?(?=regex)then|else)`
Description: If the lookahead succeeds, the "then" part must match for the overall regex to match. If the lookahead fails, the "else" part must match for the overall regex to match. Not just positive lookahead, but all four lookarounds can be used. Note that the lookahead is zero-width, so the "then" and "else" parts need to match and consume the part of the text matched by the lookahead as well.
Example: «`(?(?<=a)b|c)`» matches the second „`b`" and the first „`c`" in "`babxcac`"

Character: `(?(1)then|else)`
Description: If the first capturing group took part in the match attempt thus far, the "then" part must match for the overall regex to match. If the first capturing group did not take part in the match, the "else" part must match for the overall regex to match.
Example: «`(a)?(?(1)b|c)`» matches „`ab`", the first „`c`" and the second „`c`" in "`babxcac`"

## Comments

Character: `(?#comment)`
Description: Everything between `(?#` and `)` is ignored by the regex engine.
Example: «`a(?#foobar)b`» matches „`ab`"

# 3. Unicode Syntax Reference

## Unicode Characters

Character: `\X`
Description: Matches a single Unicode grapheme, whether encoded as a single code point or multiple code points using combining marks. A grapheme most closely resembles the everyday concept of a "character".
Example: «`\X`» matches „à" encoded as U+0061 U+0300, „à" encoded as U+00E0, „©", etc.

Character: `\uFFFF` where FFFF are 4 hexadecimal digits
Description: Matches a specific Unicode code point. Can be used inside character classes.
Example: «`\u00E0`» matches „à" encoded as U+00E0 only. «`\u00A9`» matches „©"

Character: `\x{FFFF}` where FFFF are 1 to 4 hexadecimal digits
Description: Perl syntax to match a specific Unicode code point. Can be used inside character classes.
Example: «`\x{E0}`» matches „à" encoded as U+00E0 only. «`\x{A9}`» matches „©"

## Unicode Properties, Scripts and Blocks

Character: `\p{L}` or `\p{Letter}`
Description: Matches a single Unicode code point that has the property "letter". See Unicode Character Properties in the tutorial for a complete list of properties. Each Unicode code point has exactly one property. Can be used inside character classes.
Example: «`\p{L}`» matches „à" encoded as U+00E0; «`\p{S}`» matches „©"

Character: `\p{Arabic}`
Description: Matches a single Unicode code point that is part of the Unicode script "Arabic". See Unicode Scripts in the tutorial for a complete list of scripts. Each Unicode code point is part of exactly one script. Can be used inside character classes.
Example: «`\p{Thai}`» matches one of 83 code points in Thai script, from „ก" until „๛"

Character: `\p{InBasicLatin}`
Description: Matches a single Unicode code point that is part of the Unicode block "BasicLatin". See Unicode Blocks in the tutorial for a complete list of blocks. Each Unicode code point is part of exactly one block. Blocks may contain unassigned code points. Can be used inside character classes.
Example: «`\p{InLatinExtended-A}`» any of the code points in the block U+100 until U+17F („Ā" until „ſ")

Character: `\P{L}` or `\P{Letter}`
Description: Matches a single Unicode code point that does *not* have the property "letter". You can also use \P to match a code point that is not part of a particular Unicode block or script. Can be used inside character classes.
Example: «`\P{L}`» matches „©"

# 4. Syntax Reference for Specific Regex Flavors

## .NET Syntax for Named Capture and Backreferences

Character: `(?<name>regex)`
Description: Round brackets group the regex between them. They capture the text matched by the regex inside them that can be referenced by the name between the sharp brackets. The name may consist of letters and digits.
Character: `(?'name'regex)`
Description: Round brackets group the regex between them. They capture the text matched by the regex inside them that can be referenced by the name between the single quotes. The name may consist of letters and digits.
Character: `\k<name>`
Description: Substituted with the text matched by the capturing group with the given name.
Example: «`(?<group>abc|def)=\k<group>`» matches „`abc=abc`" or „`def=def`", but not "`abc=def`" or "`def=abc`".

Character: `\k'name'`
Description: Substituted with the text matched by the capturing group with the given name.
Example: «`(?'group'abc|def)=\k'group'`» matches „`abc=abc`" or „`def=def`", but not "`abc=def`" or "`def=abc`".

Character: `(?(name)then|else)`
Description: If the capturing group "name" took part in the match attempt thus far, the "then" part must match for the overall regex to match. If the capturing group "name" did not take part in the match, the "else" part must match for the overall regex to match.
Example: «`(?<group>a)?(?(group)b|c)`» matches „`ab`", the first „`c`" and the second „`c`" in "`babxcac`"

## Python Syntax for Named Capture and Backreferences

Character: `(?P<name>regex)`
Description: Round brackets group the regex between them. They capture the text matched by the regex inside them that can be referenced by the name between the sharp brackets. The name may consist of letters and digits.
Character: `(?P=name)`
Description: Substituted with the text matched by the capturing group with the given name. Not a group, despite the syntax using round brackets.
Example: «`(?P<group>abc|def)=(?P=group)`» matches „`abc=abc`" or „`def=def`", but not "`abc=def`" or "`def=abc`".

## XML Character Classes

Character: `\i`
Description: Matches any character that may be the first character of an XML name, i.e. «`[_:A-Za-z]`».
Character: `\c`
Description: «`\c`» matches any character that may occur after the first character in an XML name, i.e. «`[-._:A-Za-z0-9]`»
Example: «`\i\c*`» matches an XML name like „`xml:schema`"

Character: `\I`
Description: Matches any character that cannot be the first character of an XML name, i.e. «`[^_:A-Za-z]`».
Character: `\C`
Description: Matches any character that cannot occur in an XML name, i.e. «`[^-._:A-Za-z0-9]`».
Character: `[abc-[xyz]]`
Description: Subtracts character class "xyz" from character class "abc". The result matches any single character that occurs in the character class "abc" but not in the character class "xyz".
Example: «`[a-z-[aeiou]]`» matches any letter that is not a vowel (i.e. a consonant).

## POSIX Bracket Expressions

Character: `[:alpha:]`
Description: Matches one character from a POSIX character class. Can only be used in a bracket expression.
Example: «`[[:digit:][:lower:]]`» matches one of „`0`" through „`9`" or „`a`" through „`z`"

Character: `[.span-ll.]`
Description: Matches a POSIX collation sequence. Can only be used in a bracket expression.
Example: «`[[.span-ll.]]`» matches „`ll`" in the Spanish locale

Character: `[=x=]`
Description: Matches a POSIX character equivalence. Can only be used in a bracket expression.
Example: «`[[=e=]]`» matches „e", „é", „è" and „ê" in the French locale

# 5. Regular Expression Flavor Comparison

The table below compares which regular expression flavors support which regex features and syntax. The features are listed in the same order as in the regular expression reference.

The comparison shows regular expression flavors rather than particular applications or programming languages implementing one of those regular expression flavors.

- JGsoft: This flavor is used by the Just Great Software products, including PowerGREP and EditPad Pro.
- .NET: This flavor is used by programming languages based on the Microsoft .NET framework versions 1.x, 2.0 or 3.x. It is generally also the regex flavor used by applications developed in these programming languages.
- Java: The regex flavor of the java.util.regex package, available in the Java 4 (JDK 1.4.x) and later. A few features were added in Java 5 (JDK 1.5.x) and Java 6 (JDK 1.6.x). It is generally also the regex flavor used by applications developed in Java.
- Perl: The regex flavor used in the Perl programming language, versions 5.6 and 5.8. Versions prior to 5.6 do not support Unicode.
- PCRE: The open source PCRE library. The feature set described here is available in PCRE 5.x and 6.x. PCRE is the regex engine used by the TPerlRegEx Delphi component and the RegularExrpessions and RegularExpressionsCore units in Delphi XE and C++Builder XE.
- ECMA (JavaScript): The regular expression syntax defined in the 3rd edition of the ECMA-262 standard, which defines the scripting language commonly known as JavaScript.
- Python: The regex flavor supported by Python's built-in `re` module.
- Ruby: The regex flavor built into the Ruby programming language.
- Tcl ARE: The regex flavor developed by Henry Spencer for the `regexp` command in Tcl 8.2 and 8.4, dubbed Advanced Regular Expressions.
- POSIX BRE: Basic Regular Expressions as defined in the IEEE POSIX standard 1003.2.
- POSIX ERE: Extended Regular Expressions as defined in the IEEE POSIX standard 1003.2.
- GNU BRE: GNU Basic Regular Expressions, which are POSIX BRE with GNU extensions, used in the GNU implementations of classic UNIX tools.
- GNU ERE: GNU Extended Regular Expressions, which are POSIX ERE with GNU extensions, used in the GNU implementations of classic UNIX tools.
- XML: The regular expression flavor defined in the XML Schema standard.
- XPath: The regular expression flavor defined in the XQuery 1.0 and XPath 2.0 Functions and Operators standard.

Applications and languages implementing one of the above flavors are:

- AceText: Version 2 and later use the JGsoft engine. Version 1 did not support regular expressions at all.
- awk: The awk UNIX tool and programming language uses POSIX ERE.
- C#: As a .NET programming language, C# can use the System.Text.RegularExpressions classes, listed as ".NET" below.
- Delphi for .NET: As a .NET programming language, the .NET version of Delphi can use the System.Text.RegularExpressions classes, listed as ".NET" below.
- Delphi for Win32: Delphi for Win32 does not have built-in regular expression support. Many free PCRE wrappers are available.

- EditPad Pro: Version 6 and later use the JGsoft engine. Earlier versions used PCRE, without Unicode support.
- egrep: The traditional UNIX egrep command uses the "POSIX ERE" flavor, though not all implementations fully adhere to the standard. Linux usually ships with the GNU implementation, which use "GNU ERE".
- grep: The traditional UNIX grep command uses the "POSIX BRE" flavor, though not all implementations fully adhere to the standard. Linux usually ships with the GNU implementation, which use "GNU BRE".
- Emacs: The GNU implementation of this classic UNIX text editor uses the "GNU ERE" flavor, except that POSIX classes, collations and equivalences are not supported.
- Java: The regex flavor of the java.util.regex package is listed as "Java" in the table below.
- JavaScript: JavaScript's regex flavor is listed as "ECMA" in the table below.
- MySQL: MySQL uses POSIX Extended Regular Expressions, listed as "POSIX ERE" in the table below.
- Oracle: Oracle Database 10g implements POSIX Extended Regular Expressions, listed as "POSIX ERE" in the table below. Oracle supports backreferences `\1` through `\9`, though these are not part of the POSIX ERE standard.
- Perl: Perl's regex flavor is listed as "Perl" in the table below.
- PHP: PHP's `ereg` functions implement the "POSIX ERE" flavor, while the `preg` functions implement the "PCRE" flavor.
- PostgreSQL: PostgreSQL 7.4 and later uses Henry Spencer's "Advanced Regular Expressions" flavor, listed as "Tcl ARE" in the table below. Earlier versions used POSIX Extended Regular Expressions, listed as POSIX ERE.
- PowerGREP: Version 3 and later use the JGsoft engine. Earlier versions used PCRE, without Unicode support.
- PowerShell: PowerShell's built-in -match and -replace operators use the .NET regex flavor. PowerShell can also use the System.Text.RegularExpressions classes directly.
- Python: Python's regex flavor is listed as "Python" in the table below.
- R: The regular expression functions in the R language for statistical programming use either the POSIX ERE flavor (default), the PCRE flavor (perl = true) or the POSIX BRE flavor (perl = false, extended = false).
- REALbasic: REALbasic's RegEx class is a wrapper around PCRE.
- RegexBuddy: Version 3 and later use a special version of the JGsoft engine that emulates all the regular expression flavors in this comparison. Version 2 supported the JGsoft regex flavor only. Version 1 used PCRE, without Unicode support.
- Ruby: Ruby's regex flavor is listed as "Ruby" in the table below.
- sed: The sed UNIX tool uses POSIX BRE. Linux usually ships with the GNU implementation, which use "GNU BRE".
- Tcl: Tcl's Advanced Regular Expression flavor, the default flavor in Tcl 8.2 and later, is listed as "Tcl ARE" in the table below. Tcl's Extended Regular Expression and Basic Regular Expression flavors are listed as "POSIX ERE" and "POSIX BRE" in the table below.
- VBScript: VBScript's RegExp object uses the same regex flavor as JavaScript, which is listed as "ECMA" in the table below.
- Visual Basic 6: Visual Basic 6 does not have built-in support for regular expressions, but can easily use the "Microsoft VBScript Regular Expressions 5.5" COM object, which implements the "ECMA" flavor listed below.
- Visual Basic.NET: As a .NET programming language, VB.NET can use the System.Text.RegularExpressions classes, listed as ".NET" below.
- wxWidgets: The wxRegEx class supports 3 flavors. wxRE_ADVANCED is the "Tcl ARE" flavor, wxRE_EXTENDED is "POSIX ERE" and wxRE_BASIC is "POSIX BRE".

- XML Schema: The XML Schema regular expression flavor is listed as "XML" in the table below.
- XPath: The regex flavor used by XPath functions is listed as "XPath" in the table below.
- XQuery: The regex flavor used by XQuery functions is listed as "XPath" in the table below.

## Characters

Feature: Backslash escapes one metacharacter
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: \Q...\E escapes a string of metacharacters
Supported by: JGsoft, Java, Perl, PCRE

Feature: `\x00` through `\xFF` (ASCII character)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

Feature: `\n` (LF), `\r` (CR) and `\t` (tab)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, XML, XPath

Feature: `\f` (form feed) and `\v` (vtab)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

Feature: `\a` (bell)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE

Feature: `\e` (escape)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby, Tcl ARE

Feature: `\b` (backspace) and `\B` (backslash)
Supported by: Tcl ARE

Feature: `\cA` through `\cZ` (control character)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Tcl ARE

Feature: `\ca` through `\cz` (control character)
Supported by: JGsoft, .NET, Perl, PCRE, JavaScript, Tcl ARE

## Character Classes or Character Sets [abc]

Feature: `[abc]` character class
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `[^abc]` negated character class
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `[a-z]` character class range
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: Hyphen in [\d-z] is a literal
Supported by: JGsoft, .NET, Java, Perl, PCRE

Feature: Hyphen in [a-\d] is a literal
Supported by: JGsoft, PCRE

Feature: Backslash escapes one character class metacharacter
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, XML, XPath

Feature: \Q...\E escapes a string of character class metacharacters
Supported by: JGsoft, Java, Perl, PCRE

Feature: `\d` shorthand for digits
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, XML, XPath

Feature: `\w` shorthand for word characters
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, GNU BRE, GNU ERE, XML, XPath

Feature: `\s` shorthand for whitespace
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, GNU BRE, GNU ERE, XML, XPath

Feature: `\D`, `\W` and `\S` shorthand negated character classes
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, GNU BRE, GNU ERE, XML, XPath

Feature: `[\b]` backspace
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

## Dot

Feature: . (dot; any character except line break)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

## Anchors

Feature: ^ (start of string/line)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XPath

Feature: `$` (end of string/line)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XPath

Feature: `\A` (start of string)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE

Feature: `\Z` (end of string, before final line break)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby, Tcl ARE

Feature: `\z` (end of string)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby

Feature: `\`` (start of string)
Supported by: GNU BRE, GNU ERE

Feature: `\'` (end of string)
Supported by: GNU BRE, GNU ERE

## Word Boundaries

Feature: `\b` (at the beginning or end of a word)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, GNU BRE, GNU ERE

Feature: `\B` (NOT at the beginning or end of a word)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, GNU BRE, GNU ERE

Feature: `\y` (at the beginning or end of a word)
Supported by: JGsoft, Tcl ARE

Feature: `\Y` (NOT at the beginning or end of a word)
Supported by: JGsoft, Tcl ARE

Feature: `\m` (at the beginning of a word)
Supported by: JGsoft, Tcl ARE

Feature: `\M` (at the end of a word)
Supported by: JGsoft, Tcl ARE

Feature: `\<` (at the beginning of a word)
Supported by: GNU BRE, GNU ERE

Feature: `\>` (at the end of a word)
Supported by: GNU BRE, GNU ERE

## Alternation

Feature: `|` (alternation)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

## Quantifiers

Feature: `?` (0 or 1)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `*` (0 or more)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `+` (1 or more)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `{n}` (exactly n)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `{n,m}` (between n and m)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `{n,}` (n or more)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `?` after any of the above quantifiers to make it "lazy"
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, XPath

## Grouping and Backreferences

Feature: `(regex)` (numbered capturing group)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE, XML, XPath

Feature: `(?:regex)` (non-capturing group)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

Feature: `\1` through `\9` (backreferences)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, POSIX BRE, GNU BRE, GNU ERE, XPath

Feature: `\10` through `\99` (backreferences)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE, XPath

Feature: Forward references `\1` through `\9`
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby

Feature: Nested references `\1` through `\9`
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Ruby

Feature: Backreferences non-existent groups are an error
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Tcl ARE, POSIX BRE, GNU BRE, GNU ERE, XPath

Feature: Backreferences to failed groups also fail
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE, POSIX BRE, GNU BRE, GNU ERE, XPath

## Modifiers

Feature: `(?i)` (case insensitive)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE

Feature: `(?s)` (dot matches newlines)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby

Feature: `(?m)` (`^` and `$` match at line breaks)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python

Feature: `(?x)` (free-spacing mode)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE

Feature: `(?n)` (explicit capture)
Supported by: JGsoft, .NET

Feature: `(?-ismxn)` (turn off mode modifiers)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby

Feature: `(?ismxn:group)` (mode modifiers local to group)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby

## Atomic Grouping and Possessive Quantifiers

Feature: `(?>regex)` (atomic group)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby

Feature: `?+`, `*+`, `++` and `{m,n}+` (possessive quantifiers)
Supported by: JGsoft, Java, PCRE

## Lookaround

Feature: `(?=regex)` (positive lookahead)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

Feature: `(?!regex)` (negative lookahead)
Supported by: JGsoft, .NET, Java, Perl, PCRE, JavaScript, Python, Ruby, Tcl ARE

Feature: `(?<=text)` (positive lookbehind)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python

Feature: `(?<!text)` (negative lookbehind)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python

## Continuing from The Previous Match

Feature: `\G` (start of match attempt)
Supported by: JGsoft, .NET, Java, Perl, PCRE, Ruby

## Conditionals

Feature: `(?(?=regex)then|else)` (using any lookaround)
Supported by: JGsoft, .NET, Perl, PCRE

Feature: `(?(regex)then|else)`
Supported by: .NET

Feature: `(?(1)then|else)`
Supported by: JGsoft, .NET, Perl, PCRE, Python

Feature: `(?(group)then|else)`
Supported by: JGsoft, .NET, PCRE, Python

## Comments

Feature: `(?#comment)`
Supported by: JGsoft, .NET, Perl, PCRE, Python, Ruby, Tcl ARE

## Free-Spacing Syntax

Feature: Free-spacing syntax supported
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE, XPath

Feature: Character class is a single token
Supported by: JGsoft, .NET, Perl, PCRE, Python, Ruby, Tcl ARE, XPath

Feature: # starts a comment
Supported by: JGsoft, .NET, Java, Perl, PCRE, Python, Ruby, Tcl ARE

## Unicode Characters

Feature: `\X` (Unicode grapheme)
Supported by: JGsoft, Perl, PCRE

Feature: `\u0000` through `\uFFFF` (Unicode character)
Supported by: JGsoft, .NET, Java, JavaScript, Python, Tcl ARE

Feature: `\x{0}` through `\x{FFFF}` (Unicode character)
Supported by: JGsoft, Perl, PCRE

## Unicode Properties, Scripts and Blocks

Feature: `\pL` through `\pC` (Unicode properties)
Supported by: JGsoft, Java, Perl, PCRE

Feature: `\p{L}` through `\p{C}` (Unicode properties)
Supported by: JGsoft, .NET, Java, Perl, PCRE, XML, XPath

Feature: `\p{Lu}` through `\p{Cn}` (Unicode property)
Supported by: JGsoft, .NET, Java, Perl, PCRE, XML, XPath

Feature: `\p{L&}` and `\p{Letter&}` (equivalent of `[\p{Lu}\p{Ll}\p{Lt}]` Unicode properties)
Supported by: JGsoft, Perl, PCRE

Feature: `\p{IsL}` through `\p{IsC}` (Unicode properties)
Supported by: JGsoft, Java, Perl

Feature: `\p{IsLu}` through `\p{IsCn}` (Unicode property)
Supported by: JGsoft, Java, Perl

Feature: `\p{Letter}` through `\p{Other}` (Unicode properties)
Supported by: JGsoft, Perl

Feature: `\p{Lowercase_Letter}` through `\p{Not_Assigned}` (Unicode property)
Supported by: JGsoft, Perl

Feature: `\p{IsLetter}` through `\p{IsOther}` (Unicode properties)
Supported by: JGsoft, Perl

Feature: `\p{IsLowercase_Letter}` through `\p{IsNot_Assigned}` (Unicode property)
Supported by: JGsoft, Perl

Feature: `\p{Arabic}` through `\p{Yi}` (Unicode script)
Supported by: JGsoft, Perl, PCRE

Feature: `\p{IsArabic}` through `\p{IsYi}` (Unicode script)
Supported by: JGsoft, Perl

Feature: `\p{BasicLatin}` through `\p{Specials}` (Unicode block)
Supported by: JGsoft, Perl

Feature: `\p{InBasicLatin}` through `\p{InSpecials}` (Unicode block)
Supported by: JGsoft, Java, Perl

Feature: `\p{IsBasicLatin}` through `\p{IsSpecials}` (Unicode block)
Supported by: JGsoft, .NET, Perl, XML, XPath

Feature: Part between `{}` in all of the above is case insensitive
Supported by: JGsoft, Perl

Feature: Spaces, hyphens and underscores allowed in all long names listed above (e.g. BasicLatin can be written as Basic-Latin or Basic_Latin or Basic Latin)
Supported by: JGsoft, Java, Perl

Feature: `\P` (negated variants of all `\p` as listed above)
Supported by: JGsoft, .NET, Java, Perl, PCRE, XML, XPath

Feature: `\p{^...}` (negated variants of all `\p{...}` as listed above)
Supported by: JGsoft, Perl, PCRE

## Named Capture and Backreferences

Feature: `(?<name>regex)` (.NET-style named capturing group)
Supported by: JGsoft, .NET

Feature: `(?'name'regex)` (.NET-style named capturing group)
Supported by: JGsoft, .NET

Feature: `\k<name>` (.NET-style named backreference)
Supported by: JGsoft, .NET

Feature: `\k'name'` (.NET-style named backreference)
Supported by: JGsoft, .NET

Feature: `(?P<name>regex)` (Python-style named capturing group
Supported by: JGsoft, PCRE, Python

Feature: `(?P=name)` (Python-style named backreference)
Supported by: JGsoft, PCRE, Python

Feature: multiple capturing groups can have the same name
Supported by: JGsoft, .NET

## XML Character Classes

Feature: `\i`, `\I`, `\c` and `\C` shorthand XML name character classes
Supported by: XML, XPath

Feature: `[abc-[abc]]` character class subtraction
Supported by: JGsoft, .NET, XML, XPath

## POSIX Bracket Expressions

Feature: `[:alpha:]` POSIX character class
Supported by: JGsoft, Perl, PCRE, Ruby, Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE

Feature: `\p{Alpha}` POSIX character class
Supported by: JGsoft, Java

Feature: `\p{IsAlpha}` POSIX character class
Supported by: JGsoft, Perl

Feature: `[.span-ll.]` POSIX collation sequence
Supported by: Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE

Feature: `[=x=]` POSIX character equivalence
Supported by: Tcl ARE, POSIX BRE, POSIX ERE, GNU BRE, GNU ERE

# 6. Replacement Text Reference

The table below compares the various tokens that the various tools and languages discussed in this book recognize in the replacement text during search-and-replace operations.

The list of replacement text flavors is not the same as the list of regular expression flavors in the regex features comparison. The reason is that the replacements are not made by the regular expression engine, but by the tool or programming library providing the search-and-replace capability. The result is that tools or languages using the same regex engine may behave differently when it comes to making replacements. E.g. The PCRE library does not provide a search-and-replace function. All tools and languages implementing PCRE use their own search-and-replace feature, which may result in differences in the replacement text syntax. So these are listed separately.

To make the table easier to read, I did group tools and languages that use the exact same replacement text syntax. The labels for the replacement text flavors are only relevant in the table below. E.g. the .NET framework does have built-in search-and-replace function in its Regex class, which is used by all tools and languages based on the .NET framework. So these are listed together under ".NET".

Note that the escape rules below only refer to the replacement text syntax. If you type the replacement text in an input box in the application you're using, or if you retrieve the replacement text from user input in the software you're developing, these are the only escape rules that apply. If you pass the replacement text as a literal string in programming language source code, you'll need to apply the language's string escape rules on top of the replacement text escape rules. E.g. for languages that require backslashes in string literals to be escaped, you'll need to use "\\1" instead of "\1" to get the first backreference.

A flavor can have four levels of support (or non-support) for a particular token:

- A "YES" in the table below indicates the token will be substituted.
- A "no" indicates the token will remain in the replacement as literal text. Note that languages that use variable interpolation in strings may still replace tokens indicated as unsupported below, if the syntax of the token corresponds with the variable interpolation syntax. E.g. in Perl, `$0` is replaced with the name of the script.
- The "string" label indicates that the syntax is supported by string literals in the language's source code. For languages like PHP that have interpolated (double quotes) and non-interpolated (single quotes) variants, you'll need to use the interpolated string style. String-level support also means that the character escape won't be interpreted for replacement text typed in by the user or read from a file.
- Finally, "error" indicates the token will result in an error condition or exception, preventing any replacements being made at all.

- JGsoft: This flavor is used by the Just Great Software products, including PowerGREP, EditPad Pro and AceText. It is also used by the TPerlRegEx Delphi component and the RegularExrpessions and RegularExpressionsCore units in Delphi XE and C++Builder XE.
- .NET: This flavor is used by programming languages based on the Microsoft .NET framework versions 1.x, 2.0 or 3.0. It is generally also the regex flavor used by applications developed in these programming languages.
- Java: The regex flavor of the java.util.regex package, available in the Java 4 (JDK 1.4.x) and later. A few features were added in Java 5 (JDK 1.5.x) and Java 6 (JDK 1.6.x). It is generally also the regex flavor used by applications developed in Java.
- Perl: The regex flavor used in the Perl programming language, as of version 5.8.

- ECMA (JavaScript): The regular expression syntax defined in the 3rd edition of the ECMA-262 standard, which defines the scripting language commonly known as JavaScript. The VBscript RegExp object, which is also commonly used in VB 6 applications uses the same implementation with the same search-and-replace features. However, VBScript and VB strings don't support \xFF and \uFFFF escapes.
- Python: The regex flavor supported by Python's built-in `re` module.
- Ruby: The regex flavor built into the Ruby programming language.
- Tcl: The regex flavor used by the `regsub` command in Tcl 8.2 and 8.4, dubbed Advanced Regular Expressions in the Tcl man pages. wxWidgets uses the same flavor.
- PHP ereg: The replacement text syntax used by the ereg_replace and eregi_replace functions in PHP.
- PHP preg: The replacement text syntax used by the preg_replace function in PHP.
- REALbasic: The replacement text syntax used by the ReplaceText property of the RegEx class in REALbasic.
- Oracle: The replacement text syntax used by the REGEXP_REPLACE function in Oracle Database 10g.
- Postgres: The replacement text syntax used by the regexp_replace function in PostgreSQL.
- XPath: The replacement text syntax used by the fn:replace function in XQuery and XPath.
- R: The replacement text syntax used by the `sub` and `gsub` functions in the R language. Though R supports three regex flavors, it has only one replacement syntax for all three.

## Syntax Using Backslashes

Feature: `\&` (whole regex match)
Supported by: JGsoft, Ruby, Postgres

Feature: `\0` (whole regex match)
Supported by: JGsoft, Ruby, Tcl, PHP ereg, PHP preg, REALbasic

Feature: `\1` through `\9` (backreference)
Supported by: JGsoft, Perl, Python, Ruby, Tcl, PHP ereg, PHP preg, REALbasic, Oracle, Postgres, R

Feature: `\10` through `\99` (backreference)
Supported by: JGsoft, Python, PHP preg, REALbasic

Feature: `\10` through `\99` treated as `\1` through `\9` (and a literal digit) if fewer than 10 groups
Supported by: JGsoft

Feature: `\g<group>` (named backreference)
Supported by: JGsoft, Python

Feature: `\`` (backtick; subject text to the left of the match)
Supported by: JGsoft, Ruby

Feature: `\'` (straight quote; subject text to the right of the match)
Supported by: JGsoft, Ruby

Feature: `\+` (highest-numbered participating group)
Supported by: JGsoft, Ruby

Feature: Backslash escapes one backslash and/or dollar
Supported by: JGsoft, Java, Perl, Python, Ruby, Tcl, PHP ereg, PHP preg, REALbasic, Oracle, Postgres, XPath, R

Feature: Unescaped backslash as literal text
Supported by: JGsoft, .NET, Perl, JavaScript, Python, Ruby, Tcl, PHP ereg, PHP preg, Oracle, Postgres

## Character Escapes

Feature: `\u0000` through `\uFFFF` (Unicode character)
Supported by: JGsoft, Java, JavaScript, Python, Tcl, R

Feature: `\x{0}` through `\x{FFFF}` (Unicode character)
Supported by: JGsoft, Perl

Feature: `\x00` through `\xFF` (ASCII character)
Supported by: JGsoft, Perl, JavaScript, Python, Tcl, PHP ereg, PHP preg, REALbasic, R

## Syntax Using Dollar Signs

Feature: `$&` (whole regex match)
Supported by: JGsoft, .NET, Perl, JavaScript, REALbasic

Feature: `$0` (whole regex match)
Supported by: JGsoft, .NET, Java, PHP preg, REALbasic, XPath

Feature: `$1` through `$9` (backreference)
Supported by: JGsoft, .NET, Java, Perl, JavaScript, PHP preg, REALbasic, XPath

Feature: `$10` through `$99` (backreference)
Supported by: JGsoft, .NET, Java, Perl, JavaScript, PHP preg, REALbasic, XPath

Feature: `$10` through `$99` treated as `$1` through `$9` (and a literal digit) if fewer than 10 groups
Supported by: JGsoft, Java, JavaScript, XPath

Feature: `${1}` through `${99}` (backreference)
Supported by: JGsoft, .NET, Perl, PHP preg

Feature: `${group}` (named backreference)
Supported by: JGsoft, .NET

Feature: `$`` (backtick; subject text to the left of the match)
Supported by: JGsoft, .NET, Perl, JavaScript, REALbasic

Feature: `$'` (straight quote; subject text to the right of the match)
Supported by: JGsoft, .NET, Perl, JavaScript, REALbasic

Feature: `$_` (entire subject string)
Supported by: JGsoft, .NET, Perl

Feature: `$+` (highest-numbered participating group)
Supported by: JGsoft, Perl

Feature: `$+` (highest-numbered group in the regex)
Supported by: .NET

Feature: `$$` (escape dollar with another dollar)
Supported by: JGsoft, .NET, JavaScript

Feature: `$` (unescaped dollar as literal text)
Supported by: JGsoft, .NET, JavaScript, Python, Ruby, Tcl, PHP ereg, PHP preg, Oracle, Postgres, R

## Tokens Without a Backslash or Dollar

Feature: & (whole regex match)
Supported by: Tcl

## General Replacement Text Behavior

Feature: Backreferences to non-existent groups are silently removed
Supported by: JGsoft, Perl, Ruby, Tcl, PHP preg, REALbasic, Oracle, Postgres, XPath

## Highest-Numbered Capturing Group

The `$+` token is listed twice, because it doesn't have the same meaning in the languages that support it. It was introduced in Perl, where the `$+` variable holds the text matched by the highest-numbered capturing group that actually participated in the match. In several languages and libraries that intended to copy this feature, such as .NET and JavaScript, `$+` is replaced with the highest-numbered capturing group, whether it participated in the match or not.

E.g. in the regex «`a(\d)|x(\w)`» the highest-numbered capturing group is the second one. When this regex matches „`a4`", the first capturing group matches „`4`", while the second group doesn't participate in the match attempt at all. In Perl, `$+` will hold the „`4`" matched by the first capturing group, which is the highest-numbered group that actually participated in the match. In .NET or JavaScript, `$+` will be substituted with nothing, since the highest-numbered group in the regex didn't capture anything. When the same regex matches „`xy`", Perl, .NET and JavaScript will all store „`y`" in `$+`.

Also note that .NET numbers named capturing groups after all non-named groups. This means that in .NET, `$+` will always be substituted with the text matched by the last named group in the regex, whether it is followed by non-named groups or not, and whether it actually participated in the match or not.

# Index


$. *see* dollar sign
[. *see* square bracket
]. *see* square bracket
\. *see* backslash
^. *see* caret
.. *see* dot
|. *see* vertical bar
?. *see* question mark
*. *see* star
+. *see* plus
(. *see* round bracket
). *see* round bracket
\t. *see* tab
\r. *see* carriage return
\n. *see* line feed
\a. *see* bell
\e. *see* escape
\f. *see* form feed
\v. *see* vertical tab
\d. *see* digit
\D. *see* digit
\w. *see* word character
\W. *see* word character
\s. *see* whitespace
\S. *see* whitespace
\`. *see* start file
\'. *see* end file
\b. *see* word boundary
\y. *see* word boundary
\m. *see* word boundary
\<. *see* word boundary
\>. *see* word boundary
{. *see* curly braces
\1. *see* backreference
\G. *see* previous match
\c. *see* control characters *or* XML names
\C. *see* control characters *or* XML names
\i. *see* XML names
\I. *see* XML names
#65535 and #xffff -> characters, 118
.bak, 173
.epp files, 26
.jgcscs, 144, 151, 153, 157
.jgfns, 154, 157
.tcl files, 196
32-bit, 235
64-bit, 235
8859, 109
adapt, 49
adapt case, 49
add active file, 31
add outside files, 32
add to project, 30
add to project unopened, 31
all files, 50
all projects, 50
alnum, 308
alpha, 308
alphabetic sort, 97
alternation, 265
anchor, 258, 287, 294, 300
any character, 256
append, 69
ASCII, 249, 308
ascii only, 200
assertion, 294
associations, 135
asterisk. *see* star
attachments, 14
auto indent, 121, 134
automatic reload, 170
autosave, 173
awk, 250
\b. *see* word boundary
back in edited files, 58
backreference, 271
  in a character class, 275
  number, 272
  repetition, 338
backreferences, 54
backslash, 248, 249
  in a character class, 252
backtick, 259
backtracking, 269, 332
backup, 14
backup files, 173, 221
begin file, 259
begin line, 258
begin selection, 66
begin string, 258
bell, 249
between matching brackets, 68
blank, 308

blank lines, 114, 115
block, 50, 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 73, 134
bom, 137
bookmark, 72, 73
Borland colors, 147
braces. *see* curly braces
bracket. *see* square bracket *or* parenthesis
bracket expressions, 307
bracket matching, 23
brackets, 57, 144
breaking long lines, 134
browser, 204
byte order marker, 137
byte value editor, 200
\c. *see* control characters *or* XML names
\C. *see* control characters *or* XML names
canonical equivalence
  Java, 286
capital letters, 109
capturing group, 271
caret, 248, 258, 287
  in a character class, 252
carriage return, 249
case, 49, 109
case conversion, 55
case insensitive, 287
case sensitive, 49
catastrophic backtracking, 332
character class, 252
  negated, 252
  negated shorthand, 254
  repeating, 255
  shorthand, 253
  special characters, 252
  subtract, 305
  XML names, 305
character equivalents, 310
character map, 198
character range, 252
character set. *see* character class
characters, 248
  ASCII, 249
  categories, 279
  control, 249
  digit, 253
  in a character class, 252
  invisible, 249
  metacharacters, 248
  non-printable, 249
  non-word, 253, 262
  special, 248
  Unicode, 249, 278
  whitespace, 253
  word, 253, 262
characters -> #65535, 117
characters -> #xffff, 118
characters -> htmlchar, 118
characters -> \uFFFF, 116
check for new version, 230
check spelling, 94, 95, 96
choice, 265
class, 252
clip collection, 196
clip editor, 197
clipboard, 18, 19, 20, 21, 76, 196
close, 15
close all, 15
close all but current, 16
close all but current project, 37
close all files, 33
close outside files, 33
close panels, 204
close project, 36
closed, 51
closed files, 28, 51
closing bracket, 281
closing quote, 281
cntrl, 308
code folding, 154
code page, 109, 137, 340, 341
code point, 279
collating sequences, 310
colors, 144, 147
column, 63, 65, 66, 122, 141
combining character, 280
*combining mark*, 278
combining multiple regexes, 265
command line, 78
command line parameters, 241
command line placeholders, 80
command line utilities, 77
comment out, 63, 64
comments, 63, 64, 65, 311, 312
commonly used files, 6, 8
compare changes, 105
compare files, 101, 103, 105
compatibility, 246
condition
  if-then-else, 302
conditions
  many in one regex, 298
configure, 77, 133

consolidate blank lines, 100
contents, 3
context menu, 237
continue
  from previous match, 300
control characters, 249, 281
convert, 109, 112, 113, 114, 115, 116, 117, 118, 119, 120
copy, 14, 19
copy append, 20
copy matches, 48
Copy Path to Clipboard, 178
copy rich text, 160
copy visible lines, 76
count, 47, 107, 108, 121
CR, 113
create portable installation, 235
CRLF, 112
cross. *see* plus
CSCS, 151, 153, 157
curly braces, 268
currency sign, 280
cursor, 163
custom layouts, 205
custom syntax coloring schemes, 144, 151, 153, 157
customize, 237
cut, 18
cut append, 19
cut matches, 48
\d. *see* digit
\D. *see* digit
dash, 281
data, 246
date, 21, 23, 160, 324
date format, 160
default text editor, 135
delete, 15, 24
delete blank lines, 99
delete duplicate lines, 97
delete folded lines, 76
delete line, 24
delete project, 36
desktop, 191
DFA engine, 250
diff, 101
difference, 101
digit, 253, 280, 308
distance, 331
dollar, 287
dollar sign, 248, 258
DOS, 109, 112, 137
dot, 49, 248, 256, 287
  misuse, 333
  newlines, 256
  vs. negated character class, 257
dot matches newline, 49
double click, 165
double quote, 249
double spacing, 114, 115
download from FTP, 214
downloading custom syntax coloring schemes, 151
drag tabs, 59
duplicate line, 24
duplicate lines, 97, 330
duplicate selection, 61
eager, 250, 265
EBCDIC, 109, 137
edit, 17, 18, 19, 20, 21, 23, 24, 57
edit clip, 197
editor, 160
EditPad Pro, 247
EditPad web site, 230
egrep, 250
else, 302
email, 12, 188
email address, 321
email as attachment, 14
enclosing mark, 280
encoding, 109, 137
end file, 259
end line, 258
end of line, 249
end selection, 67
end string, 258
engine, 246, 250
entire string, 258
environment variables, 80
epp files, 26
escape, 248, 249
  in a character class, 252
EUC, 109
euro, 340
example
  date, 324
  duplicate lines, 330
  exponential number, 320, 330
  floating point number, 320
  HTML tags, 315
  integer number, 330
  keywords, 330
  not meeting a condition, 328
  number, 330

numeric ranges, 318
prepend lines, 259
quoted string, 257
reserved words, 330
scientific number, 320, 330
trimming whitespace, 315
whole line, 328
exit, 16
expand selection, 68
explorer panel, 210
export file listing, 36
export preferences, 132
external commands, 77
extra, 94, 95, 97, 99, 100, 101, 103, 105, 106, 107, 108
favorite projects, 26
favorites, 6, 52
feedback, 235
feeds, 234
file, 4, 6, 8, 9, 12, 14, 15, 16
file associations, 135
file history, 221
file locks, 170
file mask, 134, 213
file navigator, 154, 224
file size, 170
file tabs, 176, 178
file types, 121, 133, 135
files, 28, 50
files matching a file mask, 213
files matching a regex, 213
files panel, 206
fill columns, 65
filter, 213
find first, 39
find last, 41
find next, 40
find on disk, 45
find previous, 41
fixed line lengths, 113
flash card, 235
flavor, 246
flex, 250
floating point number, 320
flow paragraphs, 114, 123
focus editor, 204
fold, 74, 75, 76, 154
fold all, 75
fold lines, 47, 74
fold unselected, 75
folders, 27, 170
font, 124, 132, 179
form feed, 249
format paragraphs, 114
formatted text, 160
forum, 230
forward in edited files, 58
free-spacing, 312
FTP panel, 214
full stop. *see* dot
go, 57, 58, 59
go to, 57, 68, 69, 72
go to matching bracket, 57
go to next bookmark, 71
go to next fold, 75
go to previous bookmark, 71
go to previous fold, 75
graph, 308
grapheme, 278
*greedy*, 267, 268
green EditPad icon, 135
group, 271
capturing, 271
in a character class, 275
named, 276
nested, 332
repetition, 338
gutter, 147
hard returns, 113
help, 229, 230, 235
hexadecimal mode, 134, 200
hide lines, 74
highlight all, 43
highlighting, 144
history, 52, 221
home key, 160
HTML tags, 315
htmlchar -> characters, 119
huge files, 170
hyphen, 281
in a character class, 252
\i. *see* XML names
\I. *see* XML names
icon for text files, 135
if-then-else, 302
ignore whitespace, 312
import file listing, 34
import preferences, 132
incremental search, 44
indent, 62, 134
indent wrapped lines, 123, 141
indentation size, 141
indentation spaces -> tabs, 115

initial caps, 109
insert date and time, 21
insert file, 69
insert page break, 24
install onto removable drive, 235
installing custom syntax coloring schemes, 151
instand find next, 44
instant find previous, 45
instant highlight, 43
integer number, 330
internet explorer, 204
invert, 52
invert case, 109
inverted line by line, 52
invisible characters, 249
ISO-8859, 109
jgcscs, 144, 151, 153, 157
jgfns, 154, 157
joint scrolling, 202
keyboard focus, 204
keyboard preferences, 183
keyboard shortcuts, 183, 225
keystroke macros, 89, 91, 92, 93
keywords, 330
KOI8, 109
lazy, 269
  better alternative, 269
leftmost match, 250
left-to-right, 124, 132
letter, 280, *see* word character
lex, 250
LF, 112
line, 51, 258
  begin, 258
  duplicate, 330
  end, 258
  not meeting a condition, 328
  prepend, 259
line break, 249, 287
line break style, 137
line breaks, 114
line breaks -> wrapping, 114
line by line, 51
line feed, 249
line lengths, 113
line numbers, 134, 141
line separator, 280
line spacing, 114, 115
line terminator, 249
lines, 24, 57, 63, 74, 76, 97, 107, 121
Linux, 112, 137
list all matches, 45
literal characers, 248
live spell check, 95
live spelling, 144
locale, 307
locking files, 170
lookahead, 294
lookaround, 294
  many conditions in one regex, 298
lookbehind, 295
  limitations, 295
loop, 51
loop automatically, 51
lower, 309
lower case, 109
lowercase, 55
lowercase letter, 280
\m. *see* word boundary
Macintosh, 113, 137
macros, 89, 91, 92, 93
macros menu, 89
mail, 12, 188
mail as attachment, 14
managed project, 28
many conditions in one regex, 298
margins, 134
mark, 72, 73, 280
match, 246
match mode, 287
match placeholders, 54
matching bracket, 23
matching brackets, 57, 144
mathematical symbol, 280
menu, 237
merge, 101
merging files, 101
metacharacters, 248
  in a character class, 252
milestones, 221
mode modifier, 287
mode span, 288
modifier, 287
modifier span, 288
mouse, 165
mouse pointer, 163
move file, 14
move project, 34
move selection, 61
move tabs, 59
mru, 4
MSIE, 204
multi-line, 287
multi-line mode, 258

multi-line search panel, 39
multiple regexes combined, 265
MySQL, 250
named group, 276
navigation, 154
navigator, 224
near, 331
negated character class, 252
negated shorthand, 254
negative lookahead, 294
negative lookbehind, 295
nested grouping, 332
Netscape, 204
new, 4
new editor, 160, 203
new page, 24
new project, 25
new version, 230
newline, 256, 287
news feeds, 234
next comparison mark, 106
next editing position, 58
next file, 58
next project, 59
NFA engine, 250
non-printable characters, 249
non-spacing mark, 280
notes, 196
number, 253, 280, 330
  backreference, 272
  exponential, 320, 330
  floating point, 320
  range, 318
  scientific, 320, 330
numbers, 121, 122, 141, 160
numeric ranges, 318
office 2003 display style, 204
offset, 57, 180
often-used files, 6, 8
once or more, 268
open, 4, 26, 27, 214
open closed files, 32
open files, 28
open files at startup, 173
opening bracket, 281
opening quote, 281
option, 265, 267, 268
options, 121, 122, 123, 124, 132, 159
or
  one character or another, 252
  one regex or another, 265
organize favorites, 6
OS X, 112, 137
other editor joint scrolling, 203
outdent, 62
outside files, 28
padding, 54, 55
page breaks, 24
pages, 24
panel preferences, 179
paragraph count, 107, 108
paragraph separator, 280
paragraph symbols, 141
parameters, 241
parentheses, 57
parenthesis. *see* round bracket
paste, 20, 109
pastebook, 196
path placeholders, 80
pattern, 53, 246
period. *see* dot
persistent selections, 160
pipe symbol. *see* vertical bar
placeholders, 54, 78
play instant macro, 92
plus, 248, 268
  possessive quantifiers, 289
pointer, 163
portable installation, 235
positive lookahead, 294
positive lookbehind, 295
POSIX, 307
possessive, 289
precedence, 265, 271
preferences, 77, 132, 133, 134, 135, 147, 159, 160, 163, 170, 173, 179, 180, 183, 188, 191, 193
prefix block, 65
prepare to search, 38
prepend lines, 259
previous comparison mark, 106
previous editing position, 57
previous file, 59
previous match, 300
previous project, 59
previously edited file, 58
print, 9, 69, 309
Procmail, 250
project, 25, 26, 27, 28, 30, 31, 32, 33, 34, 36, 37, 59, 72, 95, 107, 173
project tabs, 176, 178
projects, 50
properties
  Unicode, 279

punct, 309
punctuation, 281
quantifier
  backreference, 338
  backtracking, 269
  curly braces, 268
  greedy, 268
  group, 338
  lazy, 269
  nested, 332
  once or more, 268
  once-only, 289
  plus, 268
  possessive, 289
  question mark, 267
  reluctant, 269
  specific amount, 268
  star, 268
  ungreedy, 269
  zero or more, 268
  zero or once, 267
question mark, 248, 267
  common mistake, 320
  lazy quantifiers, 269
question marks instead of text, 109
quit, 16
quote, 249
quoted string, 257
range of characters, 252
read only, 6
record instant macro, 92
record macro, 89
record size, 123
rectangular selections, 66
recursively opening files from folders, 27
redo, 18
reflow paragraphs, 114
regex, 48, 53, 134, 213, 246
regex engine, 250
RegexBuddy, 53
regex-directed engine, 250
RegexMagic, 53
registry, 193
regular expression, 48, 246
regular expressions, 48, 53
reload files, 170
reload files from disk, 33
reload from disk, 14
reluctant, 269
removable drive, 235
remove all bookmarks, 73
remove all folding points, 76
remove closed files, 33
remove fold, 75
remove from project, 32
rename file, 14
rename project, 34
reopen files at startup, 173
reopen menu, 4
repetition
  backreference, 338
  backtracking, 269
  curly braces, 268
  greedy, 268
  group, 338
  lazy, 269
  nested, 332
  once or more, 268
  once-only, 289
  plus, 268
  possessive, 289
  question mark, 267
  reluctant, 269
  specific amount, 268
  star, 268
  ungreedy, 269
  zero or more, 268
  zero or once, 267
replace, 54
replace all, 42
replace and find next, 42
replace and find previous, 42
replace current, 42
requirements
  many in one regex, 298
reserved characters, 248
reserved characters -> xml entities, 119
restore default layout, 205
reuse
  part of the match, 271
rich text, 160
right margin, 134
right-to-left, 124, 132
ROT-13, 120
round bracket, 248, 271
RSS feeds, 234
running programs, 77
\s. *see* whitespace
\S. *see* whitespace
save, 8, 214
save all, 9
save all files in project, 26
save as, 8

save copy as, 14
save copy of project as, 34
save project, 26
sawtooth, 337
scratch pad, 196
script, 124, 132, 281
scroll wheel, 165
search, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 54
search and replace, 247
search options, 48, 49, 50, 51, 52
search pattern, 53
select all, 21
selection only, 50
selections, 61, 62, 66, 67, 68, 69, 95
send to, 191
separator, 280
session, 25, 26, 173
set any bookmark, 71
set bookmark, 73
several conditions in one regex, 298
SFTP, 214
shortcut keys, 183
shortcuts, 191, 225
shorthand character class, 253
  negated, 254
  XML names, 305
show any type of file, 213
show files of type, 213
show paragraphs, 141
show spaces, 141
show spaces and tabs, 122
side by side, 203
side panel preferences, 179
single quote, 249, 259
single spacing, 114, 115
single-line, 287
single-line mode, 256
smart home key, 160
SMTP, 188
snippet storage, 196
sort alphabetically, 97
sort tabs alphabetically, 59
space, 309
space separator, 280
spaces, 115, 122, 141
spacing combining mark, 280
special characters, 248
  in a character class, 252
  in programming languages, 249
specific amount, 268
spell check, 94, 96
spell check all, 95
spell check project, 95
spell check selection, 95
spelling, 144
split editor, 201
split hexadecimal and ascii, 200
split screen, 203
square bracket, 248, 252
squares instead of text, 109
SSL, 214
star, 248, 268
  common mistake, 320
start file, 259
start line, 258
start menu, 191
start string, 258
statistics, 107, 108
statusbar, 180
stay on top, 132
string, 246
  begin, 258
  end, 258
  matching entirely, 258
  quoted, 257
subtract character class, 305
suffix block, 65
support, 235
surrogate, 281
swap selections, 62
swap with clipboard, 21
symbol, 280
syntax coloring, 144, 151, 153, 157
syntax colors, 147
syntax highlighting, 144
tab, 249
tab size, 134, 141
tabs, 59, 115, 122, 176, 178
Tcl
  word boundaries, 263
TCL, 196
template files, 8
terminate lines, 249
text, 246
text cursor, 163
text editor, 247
text encoding, 109, 137, 340
text file icon, 135
text layout, 124
text-directed engine, 250
TextPad Collection Library, 196
time, 21, 160
tip of the day, 229

titlecase letter, 280
TLS, 214
to lowercase, 109
to uppercase, 109
toggle all folds, 76
toggle comment, 65
toggle fold, 74
toggle search panel, 39
tool placeholders, 80
toolbar, 237
tools, 77, 78, 84, 85
tools menu, 77
trim trailing whitespace, 100
trim whitespace, 100
trimming whitespace, 315
triple click, 165
tutorial, 246
two instances of editpad, 203
uFFFF -> characters, 116
uncomment block, 64
underscore, 253
undo, 17, 18
unfold all, 76
unfold lines, 74
ungreedy, 269
unicode, 109, 137
Unicode, 278
  blocks, 282
  canonical equivalence, 286
  categories, 279
  characters, 278
  code point, 279
  *combining mark*, 278
  grapheme, 278
  Java, 285
  normalization, 286
  Perl, 285
  properties, 279
  ranges, 282
  scripts, 281
unindent, 62
UNIX, 112, 137
unreadable file, 109
unselect, 68
unwrapping text, 114
update, 230
upgrade, 230
upload to FTP, 214
upper, 309
uppercase, 55, 109
uppercase letter, 280
usb stick, 235
using custom syntax coloring schemes, 153, 157
UTF-16, 137
UTF-8, 109, 137
version, 230
vertical bar, 248, 265
vertical tab, 249
view, 160, 198, 200, 201, 202, 203, 204, 205, 206, 210, 214, 221, 224
Visual Studio colors, 147
visualize line breaks, 122
visualize paragraphs, 141
visualize spaces, 122, 141
\w. *see* word character
\W. *see* word character
web browser, 204
web site of EditPad, 230
wheel, 165
whitespace, 100, 122, 253, 280, 315
  ignore, 312
whole line, 258, 328
whole word, 262, 263
whole words only, 50
Windows, 112, 137
Windows code page, 109
word, 262, 263, 309
word boundary, 262
  Tcl, 263
word character, 253, 262
word count, 107, 108
word wrap, 113, 114, 123, 134, 141
words, 50
  keywords, 330
WordStar, 186
working copies, 173
workspace, 25, 26, 173
wrapping, 114
write, 69
www, 204, 230
xdigit, 309
xml entities -> reserved characters, 119
XML names, 305
\y. *see* word boundary
zero or more, 268
zero or once, 267
zero-based line numbers, 160
zero-length match, 259
zero-width, 258, 294